ABSTRACTIONS IN
DATA BASE SYSTEM ARCHITECTURE

A thesis

presented for the Degree

of

Doctor of Philosophy in Computer Science

in the

University of Canterbury

by

M.M. Ridwan

University of Canterbury

1979

For Ceri Bryant who has endured.

Untuk Ibuku yang telah lama menunggu.

ACKNOWLEDGEMENTS

Sincere thanks are due to Dr Zen Loy, for his advice, guidance and encouragement through the greater part of this research project.

I wish to express thanks also to Dr R.E.M. Cooper, who assumed the role of supervisor during the earlier stages, and to M. Naguleswaran for her part in the programming.

Finally, I am indebted to Ceri Bryant, who has kept me in touch with the real world.

The financial assistance received from the University Grants Committee of New Zealand, in the form of a scholarship, is appreciated.

ABSTRACT

Levels of abstraction in data base system architecture are investigated. A diagrammatic representation of abstract machines in a data base system is presented.

Two major levels in a coexistence architecture: the data base modelling and data base model support levels, are considered in detail. A comprehensive treatment of abstractions in data base modelling is made and a new model, DATAM, is presented. Also presented is PRIMDAS, a primitive data structure interface for generalized model support.

Applications of PRIMDAS and DATAM are given, and details of an actual implementation are described.

INDEX OF TERMS AND ABBREVIATIONS

The section given is that at which the term or abbreviation is introduced.

Section

TABLE OF CONTENTS

Page

Page

Page

CHAPTER 1

INTRODUCTION

The importance of data base systems is reflected in the intensive research currently being carried out in data base theory and data base support. This thesis represents a further contribution to the study and construction of data base systems.

Of major concern in the data base field is the structure or framework with respect to which data base systems are constructed. As early as the 1960's (File 68 [1969], see Fry and Sibley [1976] for an historical perspective) considerations of optimization, data independence and software engineering in general have highlighted the need for multi-level data base systems. Study in data base system structuring has gathered momentum, gaining widespread recognition with the publishing of the ANSI/SPARC report in 1975 (ANSI/X3/SPARC [1975]). This report, which presents an architecture for data base systems, has since been used as the frame of reference for various refinements to the state-of-the-art (e.g. Bracchi *et al.* [1976], Klug and Tsichritzis [1977], Senko [1976b]). These studies have described how particular data base system functions can benefit from a multi-level approach.

However, the interaction of the various functional levels - levels of data representation, levels of data perception, levels of data base languages - within a single framework have not been clearly specified. It is the integration of these architectural concepts which comprises the discussion of Chapter 2. This chapter considers in detail functions and objectives in data base system architecture. A diagrammatic convention is presented, with which the architecture of particular

systems can be described.

The main impetus in the drive to more formalized data base system structuring arises from the need for individual systems to be able to support different data base models. This multiple model support is termed *coexistence* (Nijssen [1976a]). Such coexistence is seen as a compromise, where, in the final analysis, the choice of a data base model may be a matter of individual preference. In coexistence, two levels of abstraction in the architecture become significant. One is the level at which the real world is described to the data base and the other the level at which different data base models are to be supported.

1.1 DATA BASE MODELLING

At the data base modelling level, the primary consideration concerns the choice of a *community model*. The community model of a data base is the data base model with respect to which the meaning of the data base and any of its subviews are determined.

Recently (e.g. Nijssen [1976b, 1977c]), a number of data base models have been proposed as community models. Their differences lie in the concepts and the manner in which concepts of the real world are modelled. For some of these models, attempts have been made to expose and classify their similarities and differences (Kerschberg *et al.* [1976], Biller and Neuhold [1977]). No definitive conclusions have been arrived at and the question of the best model remains open. As such, new points of view and new data base models can still be seen as contributions to this area.

Any further proposals, however, need to be based on the consolidation of existing approaches. A new approach is required, since classifications evident in the literature (Kerschberg *et al.* [1976], Biller and Neuhold [1977]) have been superseded by the

introduction of further models and by the isolation of further data constructs (for example, those surveyed by Wong and Mylopoulos [1977]). Also, conceptual frameworks proposed in the literature (e.g. Smith and Smith [1977a, 1977b], Biller and Neuhold [1978]) have taken the approach of reducing abstract concepts to a few primitives which are to form the associational base of any real world modelling. This approach, although important, neglects the development of higher level abstractions of real world perception.

These more specific abstractions reflect the level of interest of the objects modelled, and of the particular form of associations perceived between them. Since such abstractions determine the behaviour of the data base, it is important that they are formalized, with their interaction and consequences completely specified.

Toward providing this consolidation and formalism, Chapter 3 presents a comprehensive treatment of abstractions of real world data modelling. Abstractions are introduced and defined within a single framework, thus providing a unifying treatment of concepts hitherto presented under varying conventions.

It is noted that in practical terms, it may not be desirable that all data base systems should support all possible abstractions. Instead, it would be more expedient to provide a range of systems supporting data base models of varying degrees of conceptual complexity and power. The capability of a system in modelling the real world is determined by the constructs that it provides. This is the basis of the data base spectrum presented in Chapter 3.

Currently lacking in the literature is a data base model which falls into the *evolutionary* category of this conceptual spectrum. An evolutionary model contains abstractions which reflect real world concepts that are subject to relatively frequent change. An evolutionary system would therefore have features enabling such

changes to be effected on the data base. Chapter 4 presents an evolutionary model based on the discussion of Chapter 3. The relationships of this model, called DATAM - Data Abstraction Model, to more traditional models are described. Also, the evolutionary operators which complement the model are given.

1.2 DATA BASE MODEL SUPPORT

Another level of abstraction that is significant in a coexistence architecture is that at which the data base models are to be supported.

Traditionally, a data base system is constructed around the data base model the system is meant to support. That is, the system is dedicated to a single model, so that support of any further models has to be done on that model's interface. Such high level support is typically cumbersome and prone to inefficiency. To avoid this, a lower level interface is required which is designed not for a specific model but allows ease in the support of different models.

With the current emphasis on data base modelling, this area of data base model support has so far received relatively little attention. Three systems which do consider the actual support of multiple models are System-R (Astrahan *et al.* [1976]), DIAM (Senko [1976b]) and LSL (Klug and Tsichritzis [1977]). The DIAM system provides a STRING model which interfaces the storage structuring facilities. This approach, however, involves access path specification and is oriented towards optimization rather than the representation of logical associations.

In System-R, network and hierarchical views are facilitated by modifying a data base based on the relational model (Codd [1970]) with the inclusion of links between tuples. LSL, a more generalized interface designed for multiple model support, takes a similar approach. Both of these approaches, however, are geared toward the support of the

traditional models: hierarchical, network and relational. Also, the interfaces are provided as high level facilities, therefore biasing the form of the interfaces that can be constructed easily on it.

To cater for a wider range of models and interfaces to these models, a more flexible support facility is required. This can be provided by a lower level interface containing generalized data structure constructs. PRIMDAS (Primitive Data Structure), described in Chapter 5, is such a facility.

PRIMDAS acts as a primitive data base system, and is used as the base machine in developing data base model interfaces. PRIMDAS is offered as a procedural constructional facility, on which specificational systems such as LSL or formal mappings (Pelagatti *et al.* [1977]) can be built. The roles of PRIMDAS in the support of a data base model, specifically DATAM, and in coexistence, are described in Chapter 6.

A PRIMDAS system has been implemented on a Burroughs B6700. It takes the form of a data sublanguage consisting of procedure calls, embedded in Burrough's Extended Algol (Burroughs Corporation [1977]). Procedures constructed on this implementation to provide a primitive DATAM system are described in Chapter 7. This chapter also describes the construction of a small data base using the DATAM system. Appendices describing particular features of the PRIMDAS interface in the current implementation, are included.

Chapter 8 presents conclusions and outlines possible directions for further work.

The architectural approach in Chapter 2 and the conceptual framework in Chapter 3 represent original work as do the DATAM and PRIMDAS models presented in Chapters 4 and 5. Chapters 6 and 7 present applications of these new concepts, and details of an actual implementation are given in the Appendices.

CHAPTER 2

DATA BASE ARCHITECTURE

2.1 INTRODUCTION

As in other large systems, data base systems can be constructed from independent but cooperating parts. For any system, its chosen decomposition depends on the particular goals and objectives demanded of the system (Ferrari [1971]).

In data base systems, the major objective is typically that of "data independence" (Date and Hopewell [1971a, 1971b], Bachman [1975], Senko *et al.* [1973]), a term which encompasses a wide range of system concerns (Engles [1970], Jardine [1973]). Regardless of the exact nature of the independence required, it is obtained by appropriate subdivision of the overall function and corresponding interfacing. This buffering technique allows particular parts of the system to be changed without impacting the rest of the system. It is these interfaces and their relationships which constitute the system's *architecture*.

An architecture is often presented as a system of abstract machines (Infotech [1977], Goos [1975]) existing at various *levels*. The level of an abstract machine corresponds to its relative distance from the base machine. Each such *level of abstraction* provides a different perception of the data base (Palmer [1974]).

It is necessary to specify the role of these levels with respect to the overall system function and in terms of the independence they achieve. Subgoals at each level also need to be specified since they determine the presentation of the abstract machine at that level.

The widespread adoption of the ANSI/SPARC guidelines (Nijssen [1976b, 1977c]) suggests that there is agreement on the general specification and function of the levels in data base architectures. Therefore, it is the subgoals or refinements which require further attention.

In the gross architecture of data base systems (Nijssen [1976a, 1977a]), it is possible to distinguish between three separate hierarchies of abstract machines, each representing distinct functions. These are: the hierarchy of data representation levels, the hierarchy of levels of data perception and the hierarchy of data base languages. Their distinction allows subgoals to become more clearly defined. Sec. 2.2 examines architectural considerations of each hierarchy and the manner in which they interrelate.

In a *coexistence architecture* (Falkenberg [1977]) the dependence of a data base system to any particular data base model is avoided. A distinction (not treated in the literature) is made in Sec. 2.3 between two forms of multiple model support. Their separate treatments provide insights into the criteria that they impose on the architecture.

In coexistence, two levels of abstraction emerge as significant: one is the level at which the semantic equivalence of data representations are determined and the other the level at which different data base models are to be supported. General criteria for these levels are outlined in Sec. 2.4.

Current diagrammatic representations of data base system architectures are not precise and do not emphasize the functional distinctions mooted in Sec. 2.2. Sec. 2.5 presents an alternative, more accurate scheme. Examples of descriptions of existing systems with these scheme are given.

2.2 HIERARCHIES OF ABSTRACT MACHINES

2.2.1 Levels of Data Representation

2.2.1.1 Four levels of abstraction

Levels in the representation of data arose from the original aim of data independence - that of gaining independence from physical storage considerations by the insertion of further layers of software between the application program and the physical data (Olle [1974b]). Application programs then no longer operate on storage representations, but rather on *logical views* of data. These logical views are data representations based on *data structures* independent of the storage structures on which they are implemented.

Further levels have been distinguished and proposed (e.g. Palmer [1974], Senko [1975], Chen [1976]) as distinct steps from the data as seen by the end-user to the data as represented on storage media. An end-user is a user that perceives data in the form available at the level furthest from the machine. Fig. 2.1 illustrates a hierarchy of levels of data representation. The data representation at each level, except for that at the lowest, is implemented on the one at the level below it. Levels 2 and 3 are the storage and logical structuring levels described previously.

A recent development is to make the data with which the bulk of users are to interact appear more meaningful (Chen [1976], Biller and Neuhold [1978], Cadiou [1976], Sundgren [1974]). This is done by representing the data in terms of objects which closely reflect data concepts of the real world. This level of abstraction is indicated as level 4 in Fig. 2.1. With the inclusion of a level to represent this *conceptual view* of data, the opportunity arises for alternative data structures to be provided for any such view (Sibley and Kerschberg [1977]).

Closer to the machine - on the bottom of the hierarchy - a further refinement is made by providing independence from the considerations of the various levels of physical storage (Senko *et al.* [1973], Olle [1974b]). Here strategies for the representation of data on a single level store - i.e. main memory - is separated from those for the representation of data in secondary storage which typically would consist of many levels (Senko and Altman [1976], Scarrott [1971], Rose and Gotterer [1973]). The two levels of abstraction required to represent this independence are shown in Fig. 2.1 as levels 2 and 1.

Functionally, this hierarchy of abstract machines represents the discrete steps in which a model of data of the real world is represented on the computer. There is no theoretical limit to the number of levels that could be inserted between the end-user and the machine, but in practical terms each further level represents a further degradation in performance.

2.2.1.2 Other approaches

Although terminologies differ, the perception of levels as found in the literature is essentially similar. Various approaches are tabulated in Figs 2.2(a) and 2.2(b).

Each row of the figures represents the levels of data representation proposed by a particular author or used in a particular system. Each row represents a progression from high level to low. The four levels described in this section are shown at the top of each table. The approximate correspondence of levels of a particular approach to those four levels is indicated by their positioning. Fig. 2.2(a) shows levels of data representation proposed as general frameworks, while Fig. 2.2(b) gives the levels of specific systems as described in the literature.

The figures show that there are fewer proposals in the lower data representation levels. This is not a reflection of the relative

importance of the levels, but of the current emphasis on conceptual data modelling aspects. It is also seen that the four levels described here correspond to the points of independence currently perceived as warranted.

2.2.1.3 The nature of the independence

This section considers the general form of the independence inherent in hierarchies of data representation levels.

Each abstract machine is implemented on the one at the level below it, so that a change in the abstract machine at a level may have repercussions on that above it. Except for the top level, the interface at each level typically cannot be independently modified. The only independent change possible is that in extending an interface by the addition of further structures or operators, perhaps as alternatives.

The major form of independence that is achieved is the freedom in the *implementation* of an interface at a level on the one below it (Senko *et al.* [1973], Frasson [1975], Hainaut [1977]). That is, an abstract machine can be reimplemented without impacting the levels above it. This is true only if no change is introduced in the abstract machine in the process.

The motivation for changing the implementation of an interface is efficiency, while the limits to which it can be done are determined by the flexibility of the interface on which it is implemented.

The discussion above concerns the dependencies of abstract machines. It describes how objects and operators provided on one level are dependent functionally on that of another. This is distinct from the dependencies of the actual data which is perceived to exist at each level.

The physical data exists only at the storage level, but each level of abstraction imposes structures representing the interpretation of the data in terms of the objects at that level. The data at each

lower level represents a rephrasing of the data in a form closer to the machine. That is, the data perceived to exist at each level are equivalent, since one can be derived from the other. This means that changes to the data at a level, e.g. deletions and additions, would have to be propagated to the data seen in other levels. Such changes are typically invoked at the top level representing a revision of the perception of the enterprise. A reimplementation of data, on the other hand, involves only a downward propagation. A reimplementation at a high level requires restructuring at lower levels (Navathe and Fry [1976], Swartout *et al.* [1977]), but the form of restructuring on lower levels should insulate higher levels. For example, if the implementation of record-occurrences of the network model is changed from sequential to linked list storage structures, changes to the physical data may be required, but the record-occurrences themselves are not altered.

That is, restructuring at a lower level occurs in two contexts: as a result of restructuring at another level or of reimplementation of a higher level.

2.2.1.4 General criteria

The distinctions above are made to clarify the nature of the independence that is achieved in the construction of levels in the representation of data. Consequently, the goals and limitations of such a construction become more well-defined. The manner in which particular systems satisfy these goals can then be used as the basis for comparison.

The following are general criteria based on the exposition given in the previous subsections.

(1) *Degree of independence* of a level - This is determined by the number of abstract machines below corresponding levels in different approaches. It represents the fineness of independent reimplementation that can be done for the abstract machine at that level.

(2) *Semantic accuracy* of the topmost level in modelling real world concepts - Although measures of this are difficult to formulate, a general rule is that architectures whose topmost level is based on some form of data structure are less accurate than those whose topmost level consists of direct representations of real world data concepts.

(3) *Flexibility* of corresponding levels - This represents the degree to which alternative abstract machines can be supported by a level.

(4) *Efficiency* of corresponding levels can also be considered - This can only be done in specific situations, involving comparable approaches and information environment.

2.2.2 Levels of Data Perception

A large data base is typically shared by many users. In such a situation it is often desirable to be able to define subsets of the data base for specific applications (Date and Hopewell [1971a], CODASYL [1971]). Reasons for this include simplicity and security (Swartout and Fry [1978]). The perception of the total data base, representing the model of the enterprise, is called the *community view* (Codd [1974]). A *subview* is a subset of a given view.

Although in practice subviewing is typically of only a single level, in concept there should be no restriction on the number of possible levels of subviewing. Therefore, in the general case, the hierarchy of levels in subviewing is as indicated in Fig. 2.3. Each view at a level (except the community view), represents a subview of a view on the level below. Therefore, along a branch in the hierarchy, there is a progressive loss of perception of the community view at each higher level.

One of the major objectives in subviewing is to insulate applications from community view changes by interfacing it to a subview

instead. Two forms of independence are required of this interfacing. The first is the independence of subviews to be able to remain static in the face of any unrelated changes on the view on which it is defined. The other subview independence is the freedom in the presentation of each view from that of any other. That is, it should be possible for subviews to be available in a form not necessarily the same as that of the view on which it is defined, so long as no extra information is introduced in the process.

This hierarchy of subview levels or *levels of data perception* is distinct from and independent of the hierarchy of levels in the representation of data (Sec. 2.2.1). As described in Sec. 2.2.1, the data contained in each data representation level are equivalent. Therefore, in principle, a view can be presented in terms of any data representation level, since functionally only the data content matters. In practice, however, the topmost data representation levels reflect the preferred ways in which to view data, so that all views and especially community views are typically presented in the forms of these levels. Also, being closer to real world concepts, subviewing at these levels is easier to formulate.

Nevertheless, architectural diagrams such as that in Fig. 2.4 are misleading, since they do not establish the independence of the concepts of representation and subviewing. It is not obvious that the lines joining external schemata to the conceptual schema and that joining the conceptual schema to the internal schema pertain to separate concepts. The alternative is to describe each hierarchy separately. A diagrammatic representation based on this scheme is given in Sec. 2.5.

Fig. 2.5 illustrates the terminologies and the representation chosen for the community view (where they are specified) for various approaches in the literature. In terms of generalized architecture, the approaches in Fig. 2.5 do not differ significantly. However, other

than the author, only one other approach (Palmer [1974]) perceives more than two levels of views.

Comparisons on the provision of subviews can be made on the basis of:

(1) The ease and scope in forming valid subviews.

(2) The degree of change in the community view that can be tolerated.

(3) The range of alternative structures/models with which to view the data.

(4) The level of subviewing provided for.

2.2.3 Levels of Data Base Languages

It is necessary to provide a range of data base languages to cater for the needs of different users (Olle [1974b], Sibley [1973], Nijssen [1976a]). Numerous data base languages are described in the literature including discussions on providing measures to determine the *level of procedurality* of data base languages (Olle [1974a], Michaels *et al.* [1976], Chamberlin [1976]).

A single data base system is to provide a set of languages of varying procedurality, giving rise to an architectural consideration involving the specifications of the interrelationships of languages or user interfaces in the system. These relationships are of a hierarchical form (Goos [1975], Kraegeloh and Lockemann [1975]). It is the levels in these hierarchies of data base languages which are considered in this section.

2.2.3.1 Language components

Data base languages consist of two components. One, a component concerned with the storage and retrieval of information in the data base, is called a *data sublanguage* (Codd [1971a]). The data sublanguage is embedded in the other component, called the *host language*, if it exists. A data sublanguage is independent of any host language it may be embedded in.

Fig. 2.6 is a diagrammatic representation of the structural relationships of languages in a data base system. The association between two related languages is that one is defined or constructed on the other. This association, however, needs further explanation in terms of the two components, host and data sublanguage.

Consider Fig. 2.7 in which a language L1 is defined on a language L2. The host language of L1, denoted as H1, need not necessarily have been derived from L2. The data sublanguage of L1, DSL1, on the other hand, is completely determined by L2. It is essentially defined on data sublanguage DSL2 of L2, since any data base operations in L1 can only be implemented by those in L2. However, in the process of construction, host facilities of L2 are typically required. The association, as abstract machines, is therefore as depicted in Fig. 2.7 where the data sublanguage of one language is defined on the totality of another language.

2.2.3.2 Relationship to levels of data representation

In a hierarchy of data representation levels (e.g. Fig. 2.1), the specification of access at one level has to be mapped to access at a lower level. This mapping may be constructed just once, but the concept of independence implies that reimplementation is possible. A language is therefore required at each level, at the very least to implement the level above it.

Besides implementation, each level of data representation forms a potential interface for user interaction. In Fig. 2.6, the set of languages depicted as being in a particular level represent those languages whose data objects are that of the corresponding data representation level. For example, assuming that the languages of Fig. 2.6 are those of the data representation levels of Fig. 2.1, then the languages{D1, D2, D3} are those based on the conceptual data representation of the system, while {C1, C2} are based on the data

structure level.

In Fig. 2.6, at each level, the association *is defined on*, is indicated by an arrow. A language at one level can either be defined on another language at the same level or on a language at the level below. However, a language at level 3, say, cannot be defined on a language at level 1, since this would violate the relationship of the levels of data representation which has its manifestations in the relationship of their interfaces.

2.2.3.3 Diagrammatic convention

At each level, the diagrammatic ordering of the languages, as in Fig. 2.6, can be made to correspond to their relative procedurality, so that there is an increasing degree of procedurality from right to left. This would indicate which languages are more procedural than others, and whether a language is defined on a procedural or a non-procedural language. This latter observation provides an estimate of the relative efficiency of an interface, since in general an interface defined on a more procedural language is more efficient than one defined on a less procedural one.

As examples of distinctive architectures, consider Figs 2.8(a) and 2.8(b). In Fig. 2.8(a), the languages can be seen to consist of two sets having a common base at the device storage level. If two copies of this common interface are used, the configuration could reasonably be seen as two separate data base systems. In Fig. 2.8(b), there is a strictly hierarchical relationship among the languages at each level, while the implementation of one level on another is through a single interface. Such a language, the most procedural in each level, is called the *base language* of that level. At each level there is a definite ordering of the procedurality and efficiency of the languages with no alternatives being provided at any point in the spectrum.

2.3 THE SUPPORT OF MULTIPLE MODELS

2.3.1 Introduction

The term *data base model* refers to a representation of data with which the real world is modelled. In a hierarchy of abstract machines in the representation of data (Fig. 2.1), the data base model is typically that at the highest level , since the objects at this level most closely reflect the data associations of the real world.

Traditionally, a data base system provides only one model. More recently, the concepts of *coexistence* (Nijssen [1976a]) or *multiple view support* (Klug and Tsichritzis [1977]) has become the accepted approach. These are typically based on the three-level framework of ANSI/SPARC (ANSI/X3/SPARC [1975]), in which a single data base is to support more than one model. This section discusses considerations in the support of multiple models, with emphasis on architectural terms.

Multiple model support can be taken to mean one of two distinct system concepts. One of these is where the system provides interfaces for different models, but any single data base can be of only one model. That is, the system provides alternative models with which a data base can be constructed but not with which to view it.

The other form of multiple model support is that of the typical interpretation, where the system allows a single data base to be viewed in terms of different models.

The considerations of each of these forms of multiple model support are discussed in Secs 2.3.2 and 2.3.3.

2.3.2 Multiple Model Support

This section considers the support of alternative models in the construction of a data base, but not necessarily in viewing it.

Consider Figs 2.9(a) and (b). These figures represent hierarchies of data representation levels in systems supporting more than

one model. Only three models are indicated in the example, but the considerations apply as well to the general case.

Each of the three models {M1, M2, M3} must eventually be implemented on the lowest data representation level. That is, there must be a sequence of abstract machines from each model to the secondary storage level. In Fig. 2.9(a), the sequences are seen to be <M1, C, B, A>, <M2, B, A> and <M3, M2, B, A>.

The prime considerations in multiple model support are the ease with which an interface of a data base model can be constructed and the efficiency of its operators. The significant factor which determines these two features is the form of the abstract machine on which the interface is directly defined.

The three models in Fig. 2.9(a) are implemented at different levels. M1 is defined on a data structure interface, M2 on a storage structure interface, while M3 is constructed on an interface to another model, M2.

Implementation at a lower level can be expected to result in a more efficient interface for two reasons. One is the decrease in the number of software levels through which the operators are mechanized, and the other is the increased control available in the use of the machine's resources. This concern over details, however, is disadvantageous, since greater effort is necessary in the construction. Specifically, the effort required in the implementation of many models on a storage structure interface is prohibitive.

On the other hand, implementation at a higher level has available objects more suited to the problem, therefore requiring less construction effort. However, besides the increased inefficiency, implementation at a very high level has a further disadvantage. Models are characterized by the differences in the objects they provide, and at this level, the objects available are specific to particular models.

The consequence is that transformations required in the construction of one model on another may be unduly awkward, e.g. as for the implementation of a network model on a strictly hierarchical one.

The remaining level, the data structure level, represents a compromise at which an interface with a suitable efficiency/ease of construction tradeoff can be provided. Fig. 2.9(b) illustrates the three models implemented on a data structure interface. The provision of such an interface, however, requires a departure from traditional data base construction, where the specifications of lower level data representations of a system are determined by the model supported. In such a system, the lower level abstract machines are only suitable for the support of that one data base model. That is, the implementation of further models cannot be easily done at these lower levels.

A data structure interface to be used as the implementation base for data base models must take into account a wide range of data models and interfaces. Sec. 2.4.2 discusses further features of such a model implementation base.

2.3.3 Coexistence

2.3.3.1 Community model

In the multiple model support of 2.3.2, where the modelling alternatives provided are in the construction of a data base, the data itself is of no consequence. Only the support of abstract machines need be considered. In coexistence systems, where users may perceive the same data with alternative representations, the major considerations involve the data.

Each data base model represents a framework with which correspondence of the data to the real world can be made. Conflicts as to what the data actually represents may therefore arise out of varying correspondence of the data to the real world. In such a

situation, a particular model (the *community model*) has to be chosen as the agreed upon representation from which all other perceptions are to be derived. That is, it is the model with which the *conceptual schema* (ANSI/X3/SPARC [1975], Bachman [1975], Biller and Neuhold [1977]) or *Common Universe of Discourse* (Nijssen [1976b, 1977c]) is constructed. This need for a special model in a coexistence environment is widely recognised (e.g. Sundgren [1974], Biller and Neuhold [1977, 1978], Nijssen [1976b]), although no widespread consensus has been reached on the specific form it should take.

The data base as seen through this special interpretation forms the *community view*. Fig. 2.10 illustrates that any alternative view relates to the enterprise only through the community view. Its perception of the associations of the enterprise and the effects of its operators are determined by transformations from the community view. Therefore, operations on the data base in terms of the other models would have to be transformed to corresponding operations based on the community model.

This leads to the two main architectural considerations in coexistence: the transformation of the data perceived in terms of the community model to that in an alternative model and the support of alternative models as abstract machines.

2.3.3.2 Schema

In the representation of an enterprise, instances of the object types available on the data base model are specified and created to represent the real world concepts. The totality of the declarations describes the universe of the objects used to represent the enterprise. This description is called the *schema* of the enterprise in terms of the model used.

In a system with many data representation levels, the objects at one level are implemented on those at a lower level. The objects

created at each level represent the same enterprise. That is, a different schema for the enterprise exists at each level. In a coexistence system, the description of each alternative view of the data represents further schemata of the enterprise in terms of the alternative models.

For clarity, the term *community schema* refers to the schema of the total data base in terms of the community model. An *alternative schema* is the schema of an alternative view, while the schema at any data representation level is appropriately prefixed, e.g. *data structure schema, storage structure schema*.

2.3.3.3 Construction of alternative perceptions

The formation of an alternative perception of the data base can now be described as the formation of an alternative schema. Consider the construction of a level 2 subview (Fig. 2.3) defined on the community view. Consider also Fig. 2.11 which illustrates architectural features of coexistence.

The schema at each data representation level can potentially be used as the basis for deriving an alternative schema. However, since correspondence to the real world is determined through the community model, derivation of an alternative schema is done most meaningfully in terms of the community schema. The formation of an alternative perception consists of a transformation from the community schema to one in the required model. In Fig. 2.11 this is indicated by the dotted line from the community view to alternative view-1.

Similarly, the construction of views at other levels consists of the derivation of a schema from that of the view on which they are defined. In Fig. 2.11 this is indicated by the dotted line from alternative view-1 to alternative view-2.

2.3.3.4 Alternative model support

The support of an alternative model, on the other hand, and as

also noted by Klug and Tsichritzis [1977], need not be implemented on a community model interface, but may be supported directly on a lower data representation level interface. Considerations in the ease of construction/efficiency tradeoff here is the same as that in the multiple model support of 2.3.2, so that a suitable implementation base would similarly be some data structure interface.

The form of the support of a community model on such a base differs from that of an alternative model. This is described below.

The meaning and growth of the physical data base is determined by the community model. This means that the hierarchy of data representation levels in the system, in fact facilitates the implementation of the community model. The schema at the implementation base represents a direct implementation of the community schema, and operations in terms of the community schema are implemented as operations on the corresponding objects at the implementation base. These two transformations need not involve any consideration of alternative views. This independent and direct support is indicated in Fig. 2.11 by the double-lined arrow.

Support of alternative models on the other hand is determined by the implementation of the community model. Here the implementation base schema of the community view is transformed to appear and behave as the alternative view. Also, the meaning of operations on the alternative view is determined by its correspondence to the community view. This means that the transformation from an alternative view to the implementation base is defined on three schemata: the alternative schema, the community schema, and its corresponding schema at the implementation base. This interpretive support is indicated in Fig. 2.11 by the single-lined arrows.

There are therefore two transformations in the support of an alternative view: schema translation (Sec. 2.3.3.3) and interpretive

support on the implementation base. Only a single set of these transformations is required for each alternative model. This corresponds to the Structural Mapping Definition of Pelagatti *et al.* [1977].

The support of an alternative model interface requires the following information:

(1) the correspondence of the community model to the alternative model,

(2) the representation of the community model on the implementation base, and

(3) the form that the interface to the alternative model is to take.

2.3.3.5 Transformations in alternative view construction

It is to be noted that alternative views are differentiated from alternative models. In particular, different alternative views may be of the same model. This is made clear by considering the general construction of alternative data perceptions as illustrated in Fig. 2.12.

The construction involves two separate processes (Nijssen [1976a]):

(1) a subviewing in terms of the community model itself.

(2) the transformation of the resultant subview into a corresponding view in terms of the desired data base model.

Operators to perform the subviewing of (1) is considered a community model related facility and is discussed in Chapter 4. The results of a subviewing is a set of community model data objects representing the required semantic subset of the community view. The description/specification of the objects in any particular subview is called a *sub-schema*. The transformation of (2) is based on some sub-schema and results in an alternative schema.

The schema translation and interface support of alternative views other than those defined directly on the community view involves further considerations. Consider alternative view-2 of Fig. 2.11 and where the models of the views in the figure are all different. There are two possible approaches in the construction of view-2.

The first is where the objects at view-2 are derived from the objects at view-1 in terms of the model at view-1. This means that the support of view-2 on the data structure level involves two levels of interpretation, one with respect to the view-2 model and the other the community model. Besides requiring numerous and complex support procedures, this approach may introduce inconsistencies through the multi-level translation of objects from one model to another.

The alternative approach and that adopted here is where all views are derived from schemata in terms of the community model. Here view-2 is constructed from the *sub-schema* at that view. With this scheme, there can be at most one level of inter-model translation, so that consistency and simplicity can be better preserved. Fig. 2.11 is to be interpreted in terms of this approach.

2.3.3.6 View dependencies

Alternative views are not totally independent from the community view. In particular, a subview and therefore also its corresponding alternative view, must at all times be a subset of the community view. This means that changes of the community schema have to be propagated through to any alternative schema which may be affected (Fig. 2.13).

The reverse, where changes to an alternative schema are to be reflected in the community schema, involves considerations of the overall use of the data base and may not always be desirable. The data base represents a central resource of data servicing a community of users and any changes, including additions to the data base, have to be carefully controlled. Where it is to be done, the changes would have

to be transformed to community model terms.

2.4 TWO MAJOR LEVELS OF ABSTRACTION

2.4.1 Community Models

Studies on data base models (Kerschberg *et al.* [1976], Biller and Neuhold [1977]) have concentrated on classifying the models on their approaches in the representation of real world concepts. Few direct evaluations (Bracchi *et al.* [1976]) have been made; while, because of the fast growing nature of the field, any such comparisons become quickly superseded. Consequently, no general consensus has been reached on which model would serve best as a community model. This is evidenced by the continuing contributions and proposals in data base modelling (e.g. Falkenberg [1977], Pirotte [1977]).

Nevertheless, preliminary analyses can still be carried out usefully by considering a model's potential in specific functions of community models. This subsection briefly discusses features considered essential for a community model and introduces a new data base model to be described in Chapter 4.

2.4.1.1 Data models and abstract models

The community model forms the semantic reference with respect to which a common interpretation of the meaning of the data base is made. As such the perception provided should closely correspond to the real world.

With this criterion, certain types of data base models become less suitable as community models compared to others. Data base models can be differentiated on their level of data representation. A *data model* is one which provides structural representations for the instances of the data associations of the real world, while an *abstract model* is one which models the enterprise view with objects which are direct

representations of distinct real world concepts (Schmid [1977] presents a similar distinction). Data models are at the data structure level while abstract models are at the conceptual level in the hierarchy of data representation levels (Fig. 2.1). The meaning attached to data described in terms of data models (e.g. network model) are open to interpretation, so that in the formation of subviews or translations to other models, intuition may be involved (Biller and Neuhold [1978]). Therefore, abstract models, in which the objects represent formal concepts, are better candidates for community models.

2.4.1.2 Criteria for abstract models

A minimum set of requirements for abstract models in a coexistence architecture, representing the consensus of the September 1976 meeting of IFIP Working Group 2.6 (Biller and Neuhold [1977], Nijssen [1977b]), are:

1. complete
2. formal
3. easy to formulate and understand
4. easy to change

Nijssen [1977b] has added three more:

5. easy to transform (to other views)
6. unique
7. orthogonal ,

while Biller and Neuhold [1977] include:

8. stable
9. form the basis for different views.

To gauge the relative importance of these criteria in the choice of a community model, consider the two main processes involved in the construction of alternative views described in Sec. 2.3.3 and illustrated in Fig. 2.12.

The deciding criteria in the choice of a community model are the ease in the formulation and understanding of community views and ease in the formulation of subviews. Difficulty in the formulation or understanding of a community view would result in ambiguity in the construction of transformations to other models. Since the specification of an alternative view involves a subsetting of the community view, it is important that rules in subviewing be simple and easy to understand.

The other requirements, including the additional one of conceptual accuracy/expressiveness, are desired features of data base models in general and do contribute to their suitability as community models. However, most models are claimed to satisfy these criteria, so that amongst those that are "good" abstract models, it is those that satisfy the two criteria above that determine the choice.

The concept of simplicity is difficult to quantify (Kent [1977]). However, the form of the constructs provided by models generally determine the ease with which they can be applied. Models such as DIAM (Senko *et al.* [1973]), Infological Model (Langefors [1974]), and Information Management Concepts (Durchholz and Richter [1974]) are based on the specification of detailed units of data associations and are correspondingly difficult to follow and utilize. It seems more natural and therefore simpler to model the real world in terms of characteristic rather than individual data associations. This, as well as other considerations of simplicity, are incorporated in the Data Abstraction Model (DATAM) presented in Chapter 4.

The final choice may be determined by individual preference. Nevertheless, the choice would be made from models having comparable semantic power based on the abstractions each provides. Towards providing a measure for this, Chapter 3 presents a comprehensive exposition of formal abstractions in the modelling of the real world

as well as a conceptual spectrum for data base models.

2.4.2 Implementation Bases

Approaches in implementation are typically specific to data base models and among these the major thrust has been in the implementation of the relational model (Senko [1977a]). Examples of these are ZETA (Mylopoulos *et al.* [1975]), INGRES (Held *et al.* [1975], System R (Astrahan *et al.* [1976]), Gamma-0 (Bjorner *et al.* [1973]).

More general approaches (Sibley and Taylor [1973], CODASYL [1977], Stocker [1977]) are oriented to the formalization of storage structuring. These approaches can be described as involved in the specification of the *mapping* or *encoding* of logical structures to storage, rather than the provision of the logical structures themselves. This is essentially also the function of the *string model* of DIAM, which has been shown to facilitate the mapping of the access paths of the Entity-Set Model (Senko *et al.* [1973]) and the relational model (Schneider [1976]) to the storage encoding level. The string model also forms the basis for access path optimization (Ghosh and Astrahan [1974]).

These approaches differ from the implementation base to be considered in the support of multiple models. The interface required here is one which provides data structure building blocks with which the instances of the data associations of the conceptual level can be perceived and operated on. Such an abstract machine would form the base on which the underlying structures and operations of data models can be constructed.

Two approaches providing for such support are System-R and the *Link and Selector* facility (Tsichritzis [1976]). However, as indicated in Chapter 1, they are mainly oriented for the support of the hierarchical, relational and network models (e.g. Klug and Tsichritzis [1977]). Further, they are offered as high level facilities. The

support of procedural interfaces would therefore encounter the awkwardness commented on by Tsichritzis (Discussion on "Views on Data", Tsichritzis and Lochovsky [1976]):

> "If you take one data model and impose another data model on it, you will find a very weird situation whereby you do something underneath, then you cover it up with something else, and then you unearth it higher up."

Also, their record-oriented approaches predetermine access so that for example, the column-oriented access of SQUARE (Boyce *et al.* [1974]) cannot be directly represented.

To provide a corresponding facility for a wider range of data base models and interfaces to them, two approaches can be taken:

(1) provide a wider range of options at a high level to cope with perceived conditions, or

(2) provide a lower level facility maximizing flexibility while retaining storage independence; in other words, avoiding initial structural or access bias.

The first approach involves providing specific facilities for each known variation. The second, more general approach is adopted here, where in Chapters 5 and 6, an interface to a primitive data structure (PRIMDAS) is presented.

2.5 DESCRIPTION OF DATA BASE SYSTEM ARCHITECTURES

2.5.1 Diagrammatic Representations

The difficulty of vigorous comparisons in data base systems has been noted (Hardgrave and Sibley [1975]). The problem is attributed to "the lack of an accepted formalism for describing and comparing data base requirements and data base systems". Data base complexity is further compounded by the advent of multi-level systems. Attempts at data base model classifications (Kerschberg *et al.* [1976], Biller and

Neuhold [1977]) are steps toward a formalism. A specification of data base architecture would contribute further to data base comparisons.

It is established in Sec. 2.2.2 that architectural descriptions of the form given in Fig. 2.14 (from Machgeels [1976]) are misleading, since they do not distinguish the functional differences of the levels. It gives no indication of the relationships of the languages. The diagram also gives the impression that particular users are associated to a single language as well as to a single alternative view. In the general case, a single user may use more than one language and interact with more than one alternative view. That is, the assignment of users to views and languages can be considered separately from the system architecture.

The main function of the diagram in Fig. 2.14 appears to illustrate the relationships of the schemata rather than that of the abstract machines of the system. Considerations in this latter, more general system aspect, however, are emphasized as the complexity of systems grow. In complex systems there may be alternatives in the levels of data representation as well as in language implementations. The description format illustrated in the next section caters for this anticipated complexity.

The format displays each of the three hierarchies of abstract machines (Sec. 2.2) of the system separately. It is a diagrammatic representation which could be augmented by some form of written description.

In a single system, if both forms of the multiple model support discussed in Sec. 2.3 are catered for, then the levels of data representation in each form also need to be described separately. This is necessary to indicate that the transformations involved in each hierarchy are actually different. Also, separate hierarchies of data base languages may be associated with each form.

2.5.2 An Illustration

Consider the hierarchies of abstract machines in the hypothetical system given in Fig. 2.15. The four levels are assumed to correspond to that of Fig. 2.1. The independent multiple model support and coexistence support are described in Figs 2.15(a) and (b) respectively. To facilitate the description of its languages, however, these hierarchies are depicted as in Fig. 2.16.

This figure consists of three parts. Part A represents the independent multiple model support, where the data representation hierarchy is given on the left in a linearized (postorder) fashion. The languages corresponding to each data representation form, and their relationships, are given on the right. For example, languages {L1, L2, L3} are on the conceptual data representation M1, while of these, L1 is defined in language L6 of the data structure representation D1.

Part B.I is the corresponding description for the coexistence model support. Although some of the languages are the same as in Part A, the translation of a language at one representation level to that of another is necessarily different (except for those of community models). The community model, M4, in the hierarchy is indicated by the double-lined box. The implementation sequence of M4, <M4, D3, S1, P1>, represented by the language sequence <L13, L15, L16, L17> may be identically that sequence in Part A. In fact, each of the independently supported models in A has the architectural potential of being a community model.

The relationships of the alternative views defined on the coexistence hierarchy in Part B.I at a particular time are given in Fig. 2.15(c) and depicted in a linear fashion in the left section of Part B.II of Fig. 2.16. For each view, its description includes the model with which it is perceived and the set of languages available

on that view. These are indicated in the right section of Part B.II. It records, for instance, that view V3 is in terms of model M1 and has available languages L1 and L2 of that model.

2.5.3 Examples

Fig. 2.17 gives examples of the architectural description of four systems from the literature. The systems described are ZETA/TORUS (Mylopoulos *et al.* [1975]), System R (Astrahan *et al.* [1976]), EDBS of the University of Toronto (Klug and Tsichritzis [1977]) and DIAM II (Senko [1976a, 1976b]). They represent those few system descriptions in the literature which include a multi-level architectural specification.

Since the literature presents only general descriptions of the systems, no information concerning specific alternative views is available and therefore Part B.II is omitted.

In ZETA/TORUS (Fig. 2.17(a)), IRL and PRL are shown at lower data representation levels reflecting their functions as interfaces to data structure and storage structure facilities. However, strictly speaking, both IRL and PRL represent interfaces of the relational model data representation. Therefore a more accurate representation is that given in Fig. 2.18. This approach is also taken for System R (Fig. 2.17(b)).

Except for DIAM II (Fig. 2.17(c)), the other systems have no clear lower level data representations. This is indicated by the dotted boxes. The literature describes ZETA/TORUS as a single model (relational) system, while the others include considerations of coexistence. The community model of EDBS (Fig. 2.17(d)) is not specified in the literature and is therefore not indicated in the figure.

Finally, a representation in this new format for the traditional

representation of Fig. 2.14 is given in Fig. 2.19. Subviews V1 and V2 correspond to the views for user 1 and user 4 of Fig. 2.14, while subviews V3 and V4 are hypothetical, based on the available models and languages. Also, the languages with which the hierarchy of data representation levels is implemented are not specified and are indicated in the figure by the dotted boxes.

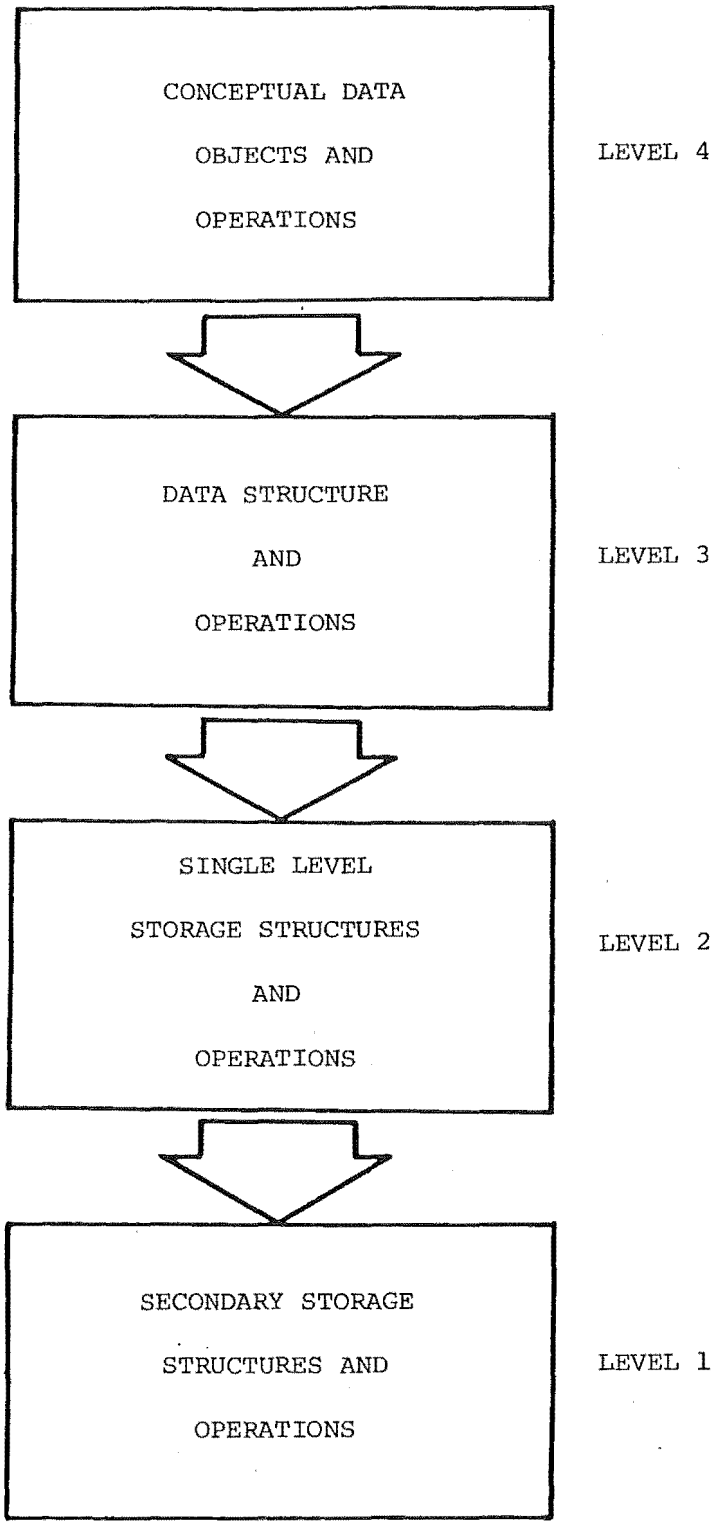


FIGURE 2.1 Hierarchy of abstract machines in the representation of data.

(a)

AUTHOR	CONCEPTUAL	DATA STRUCTURE	STORAGE STRUCTURE	SECONDARY STORAGE
ANSI/X3/SPARC [1975]	Conceptual schema		Internal schema	
Biller and Neuhold [1977]	Abstract model		Description of stored data	
Cardenas [1975]		Logical	Logical-physical	Physical organization / Data accessing
Chen [1976]	Information concerning entities and relationships which exist in our mind	Information structure	Access-path - independent data structure	Access-path - dependent data structure
Hawryszkiewycz [1978]	Information structure of the real world	Representation of information structure by data structures in the file organizations	Implementation of logic structures in physical store	
Jardine [1973]		System known logical relations	Stored data relations	
Langefors [1974]	Infological model		Datalogical model	
Nijssen [1974]	Information structure	Data structure	Storage structure	
Schmid [1977]	Unstructured conceptual model / Conceptual information structure model	Conceptual data model		
Sibley [1973]	Conceptual schema	Logical schema		
Sibley and Kerschberg [1977]	Data model theory	Data model		

(b)

SYSTEM/AUTHOR	CONCEPTUAL	DATA STRUCTURE	STORAGE STRUCTURE	SECONDARY STORAGE
Bracchi *et al.* [1974]		Binary relation	Schema file subset	System physical file
DIAM I Senko *et al.* [1973]	Entity set model	String model	Encoding model	Physical device model
Omega Schmid and Bernstein [1976]		Relational model	Operations on virtual files and data structures	Operations on physical files
Ridwan	DATAM	PRIMDAS		
System R Astrahan *et al.* [1976]		Relational data system	Relational storage system	
ZETA/TORUS Mylopoulos *et al.* [1975]		Relational model	Intermediate relational level	Primitive relational level

FIGURE 2.2 Levels of data representation.

(a) General frameworks

(b) Specific architectures

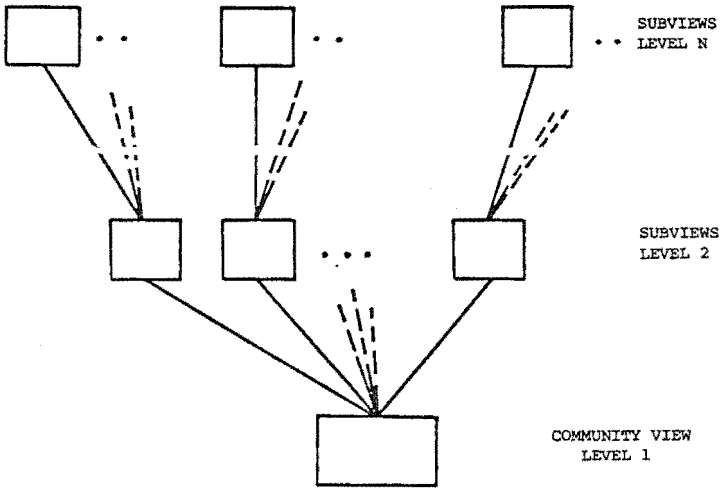


FIGURE 2.3 General hierarchy of subviewing

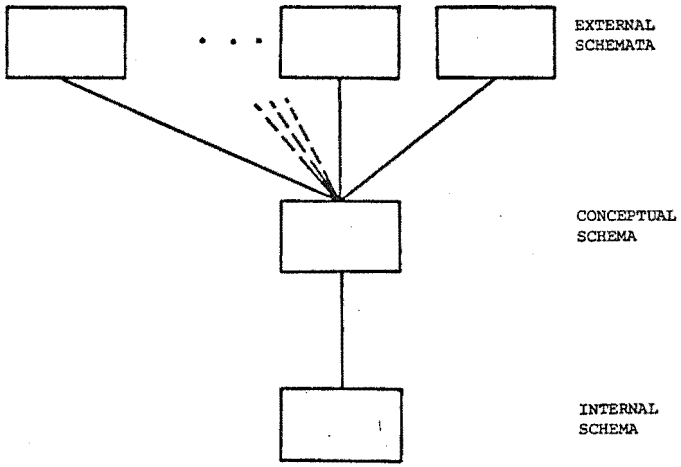


FIGURE 2.4 Typical presentation of data base system architecture

REFERENCE	SUBVIEW	COMMUNITY VIEW	NUMBER OF LEVELS PERCEIVED	REPRESENTATION OF COMMUNITY VIEW
ANSI/X3/SPARC [1975]	external model	conceptual model	2	
Bracchi *et al.* [1974]	user subschema	schema	2	binary relations
Codd [1974]	application view	community view	2	relational model
Codd and Date [1974]	problem-oriented schema	principal schema	2	relational model
Date and Hopewell [1971a]	subschema	schema	2	third normal form
Jardine [1973]	program-defined logical relations	system-known logical relations	2	
Palmer [1974]	program/ application subschema	schema	3	
Ridwan	subview	community view	n	DATAM
Senko [1976a]	end-user level	infological level	2	binary logical model
Sibley [1973]	virtual schema	conceptual schema	2	
Sundgren [1974]	filter	schema	2	e-messages

FIGURE 2.5 Approaches in subviewing

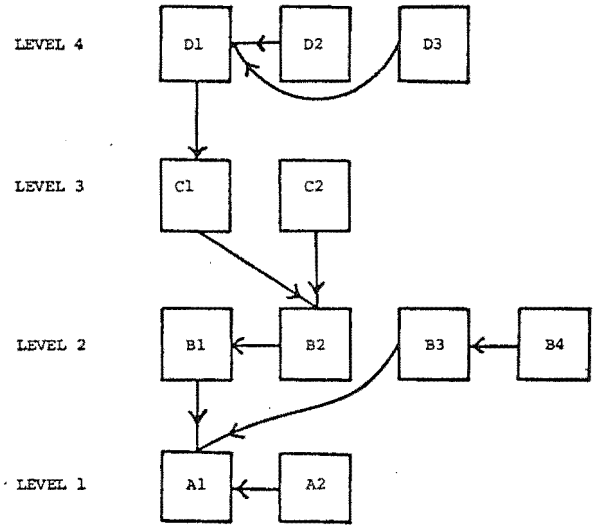


FIGURE 2.6 Representation of language support

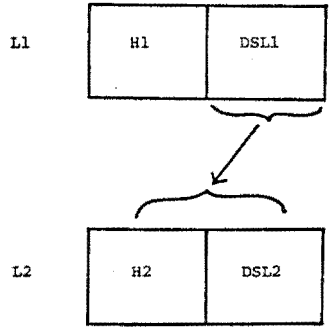
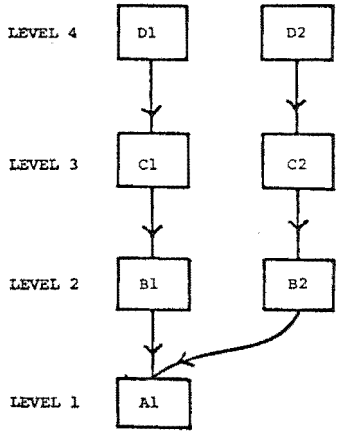


FIGURE 2.7 Language implementation

(a)



(b)

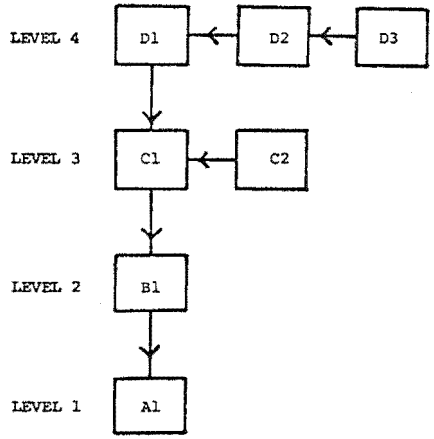
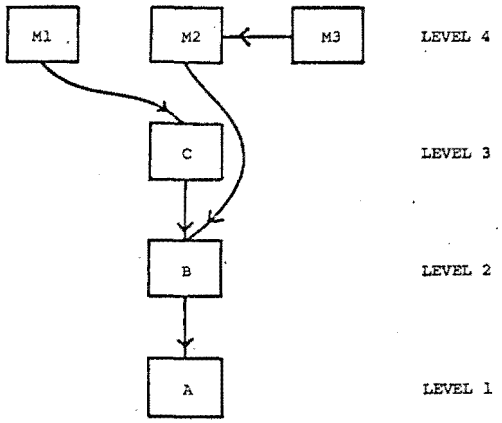


FIGURE 2.8 Hierarchies of data base interfaces

(a)



(b)

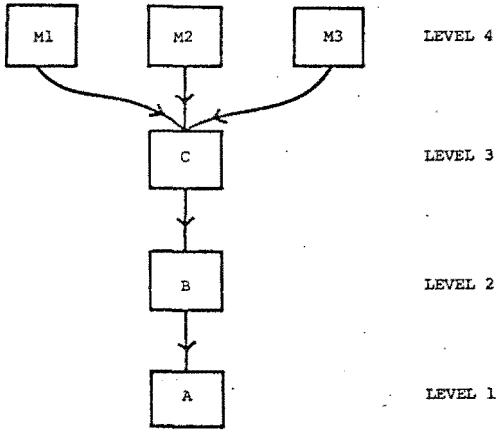


FIGURE 2.9 Hierarchies in multiple model support

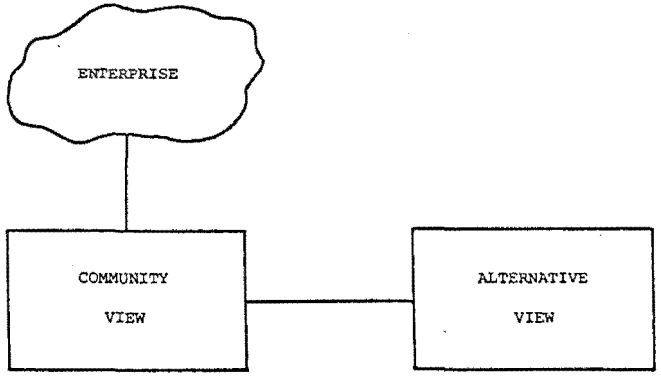


FIGURE 2.10 Alternative viewing

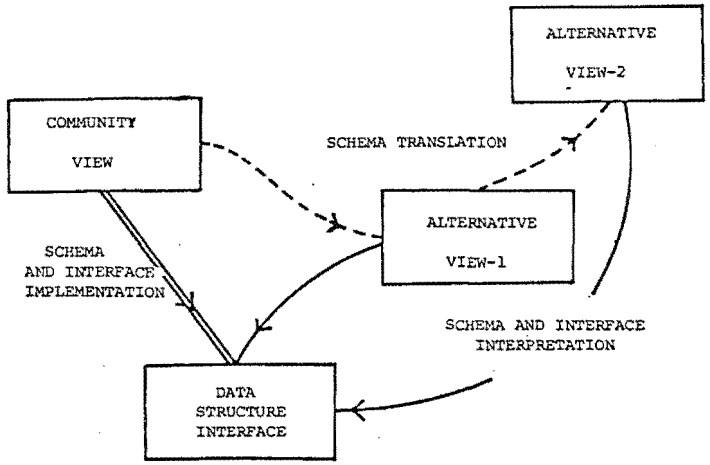


FIGURE 2.11 Support of multiple model viewing

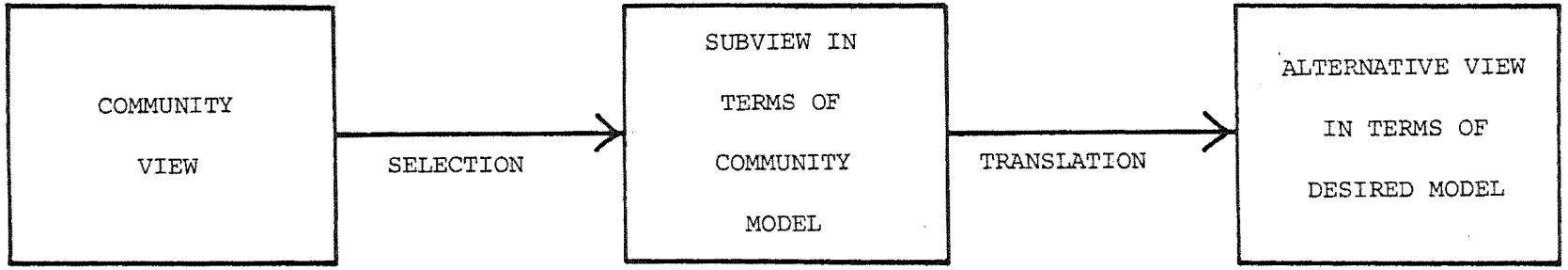


FIGURE 2.12 Construction of alternative data perceptions

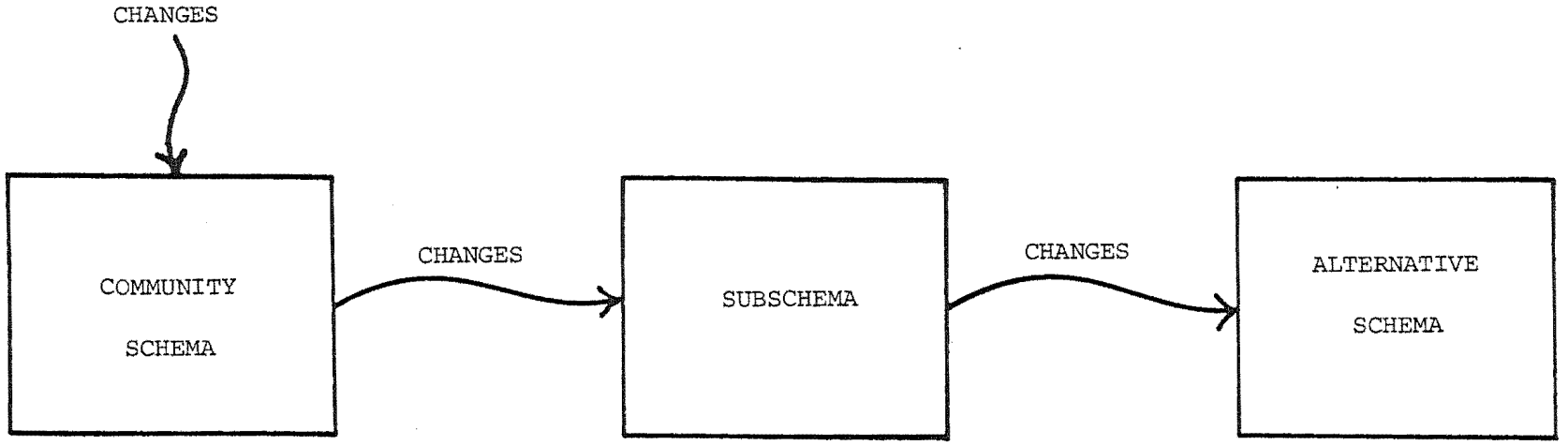


FIGURE 2.13 Propagation of view changes

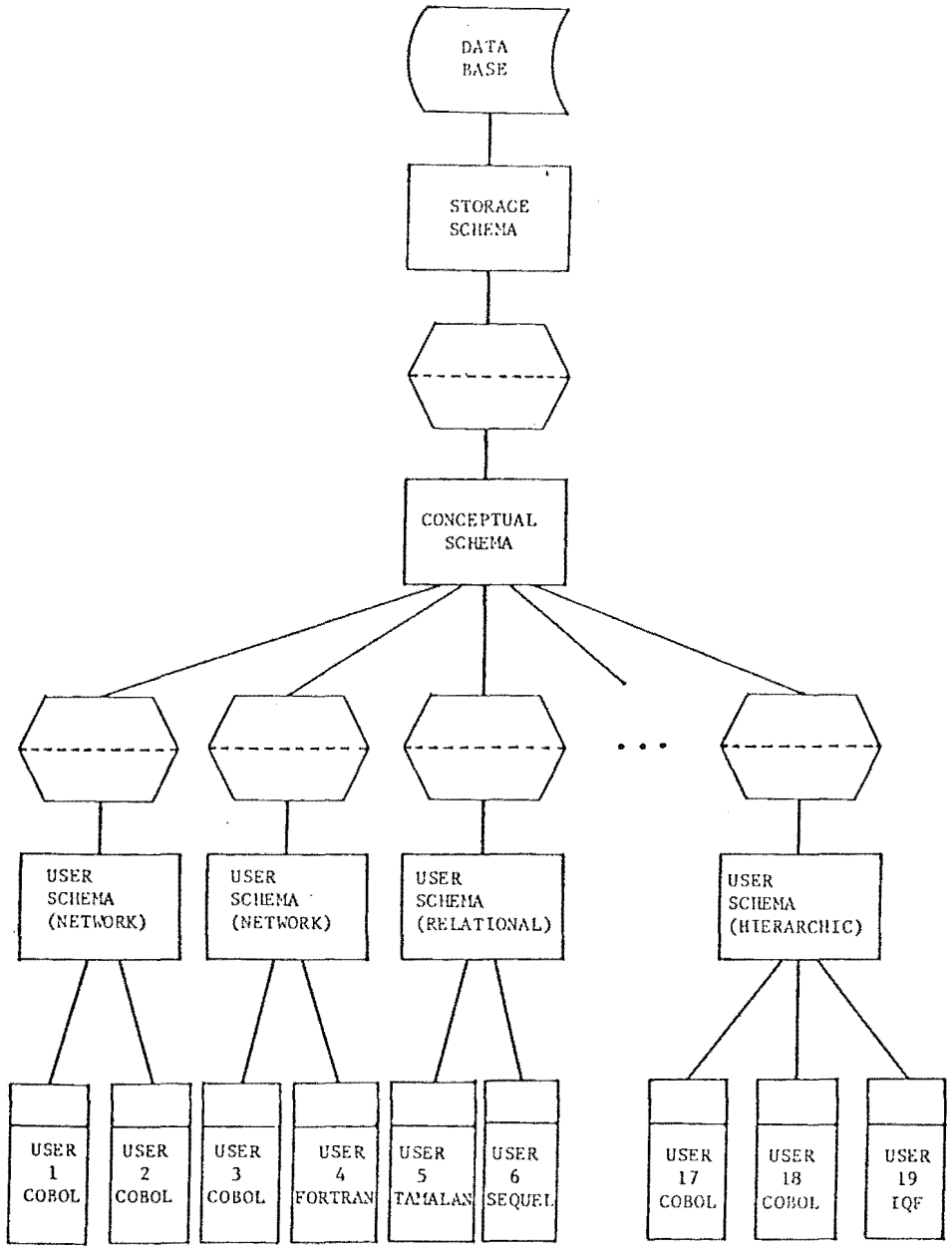
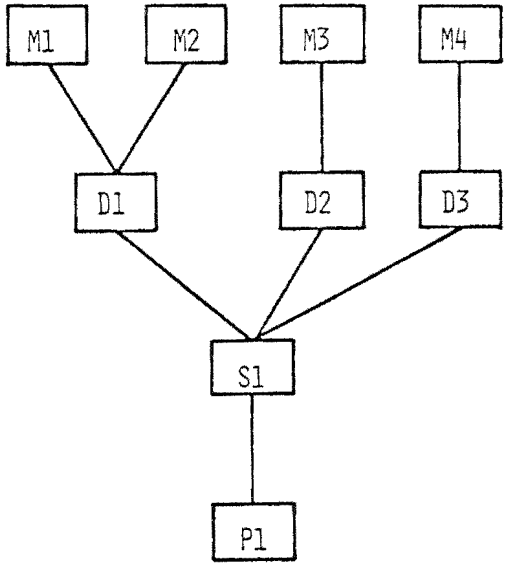
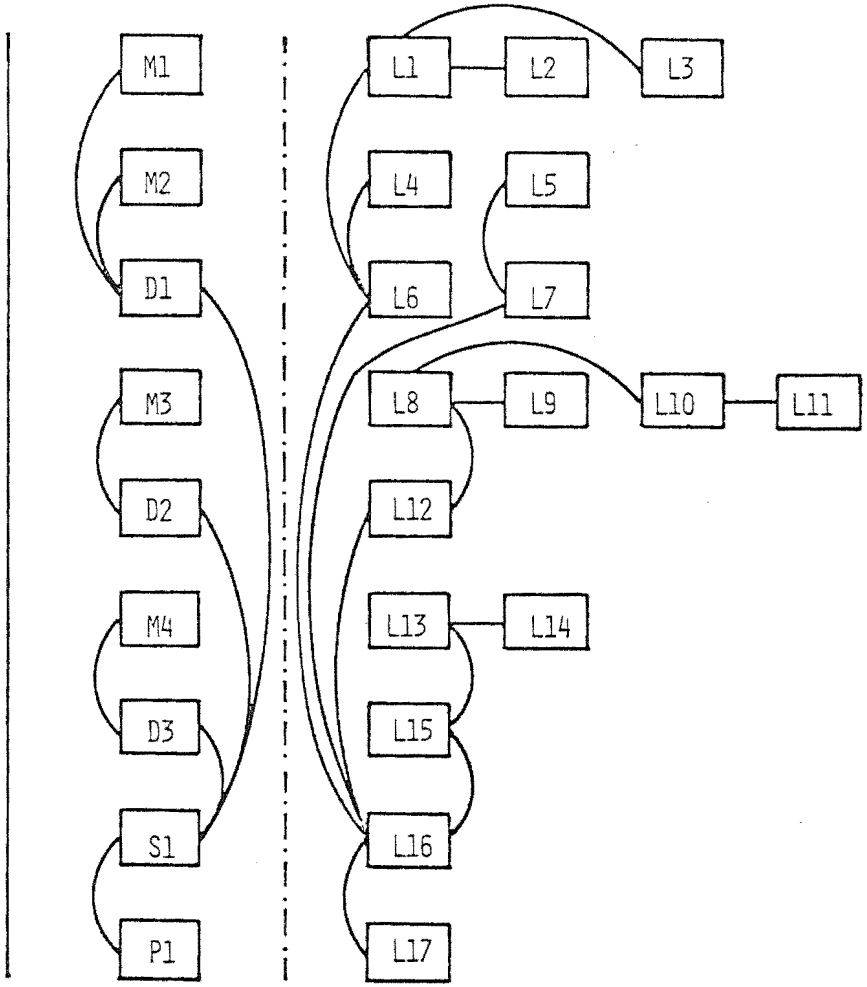


FIGURE 2.14 Traditional architectural representation

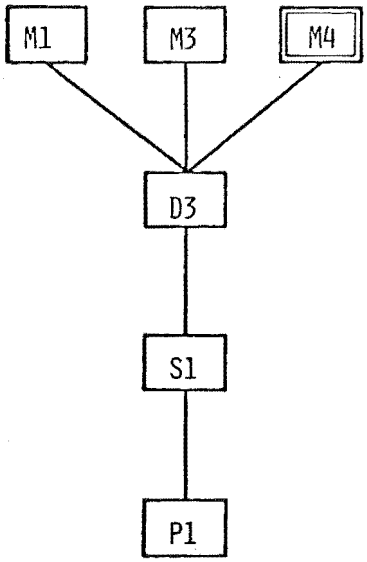
A. INDEPENDENT MODEL SUPPORT

(a)



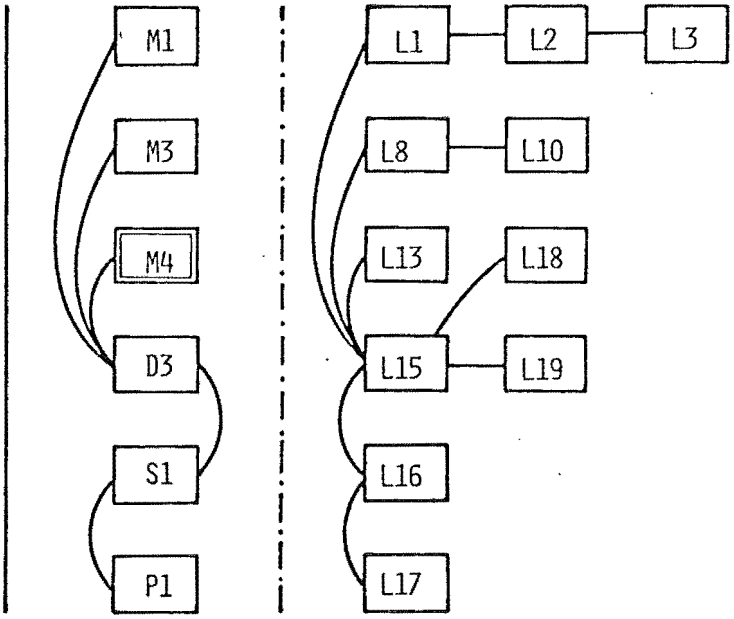


(b)

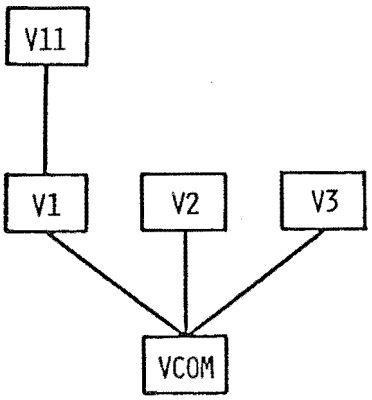


B. COEXISTENCE

I. MODEL SUPPORT



(c)



II. DATA PERCEPTION HIERARCHY

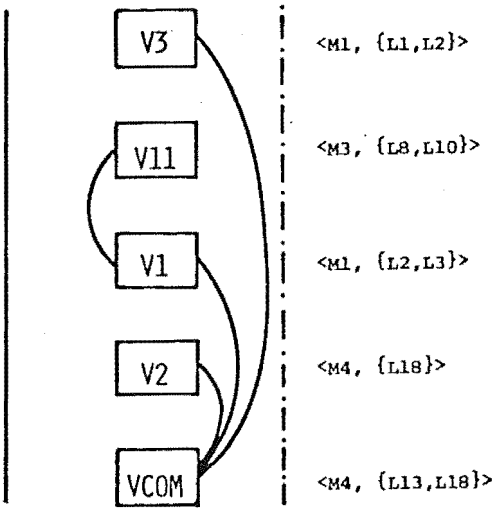


FIGURE 2.15 Hierarchies of abstract machines in a hypothetical system.

(a) Independent model support

(b) Coexistence model support

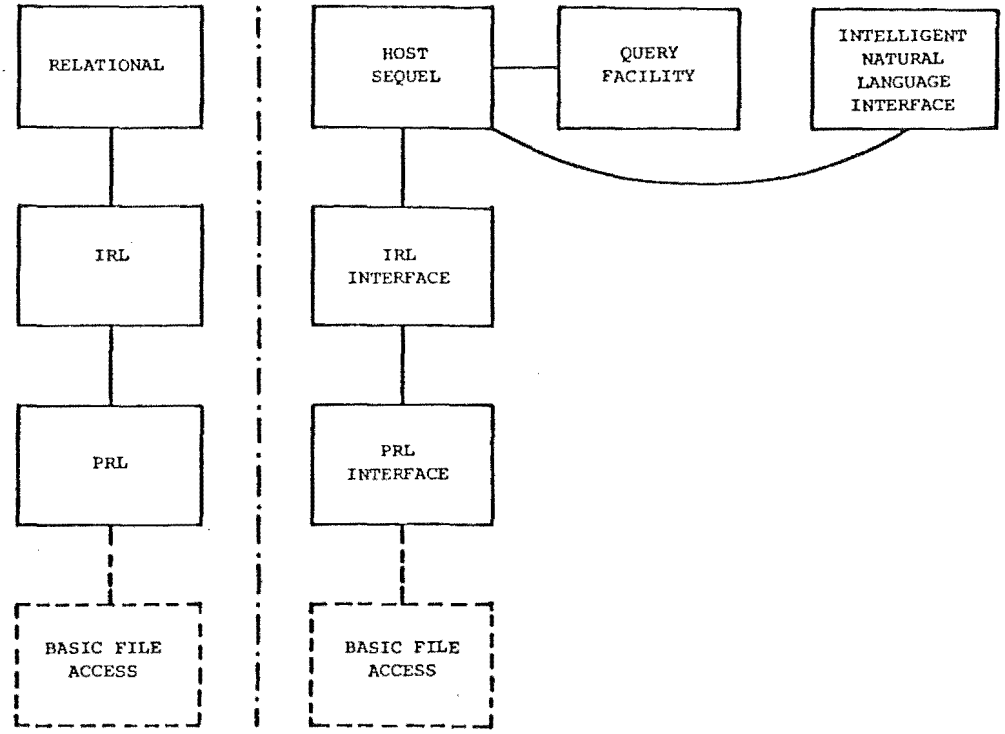
(c) Levels of data perception

FIGURE 2.16 Representation of Figure 2.15

(a)

ZETA/TORUS:

A. INDEPENDENT MODEL SUPPORT



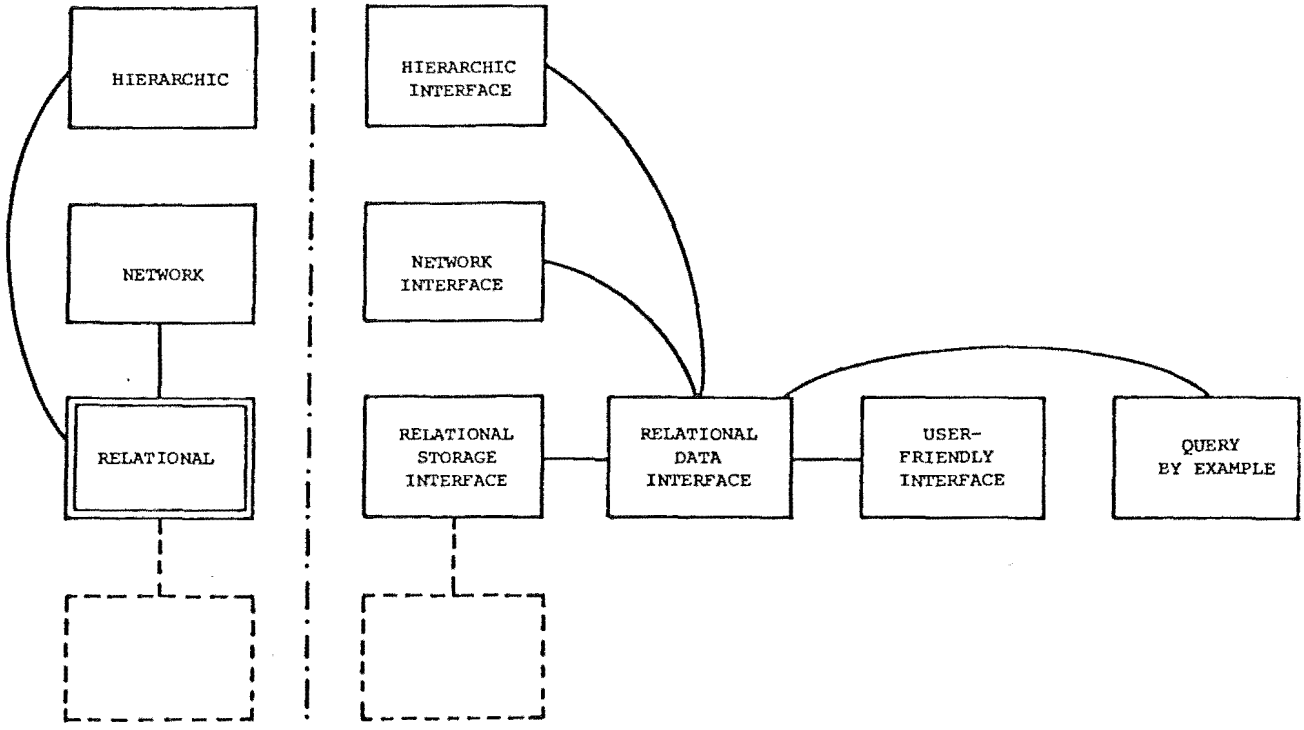
PRL - Primitive Relational Level
IRL - Intermediate Relational Level
HOST SEQUEL is SEQUEL embedded in PL/1

(b)

SYSTEM R:

B. COEXISTENCE

I. MODEL SUPPORT

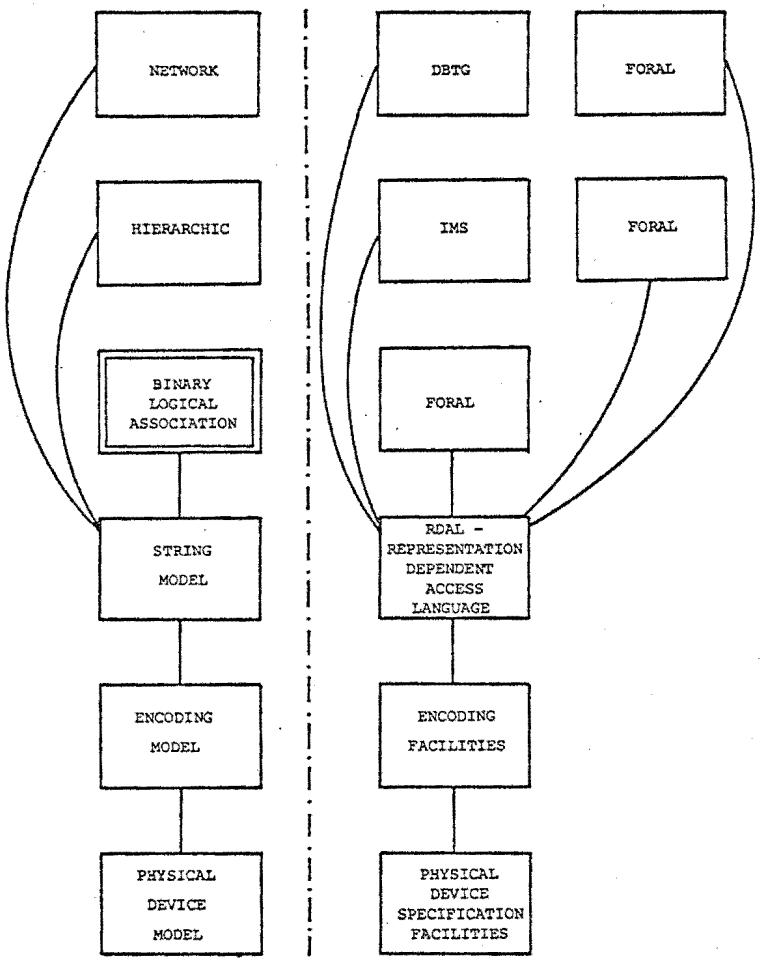


(c)

DIAM II:

B. COEXISTENCE

I. MODEL SUPPORT



(d)

EDBS (UNIVERSITY OF TORONTO):

B. COEXISTENCE

I. MODEL SUPPORT

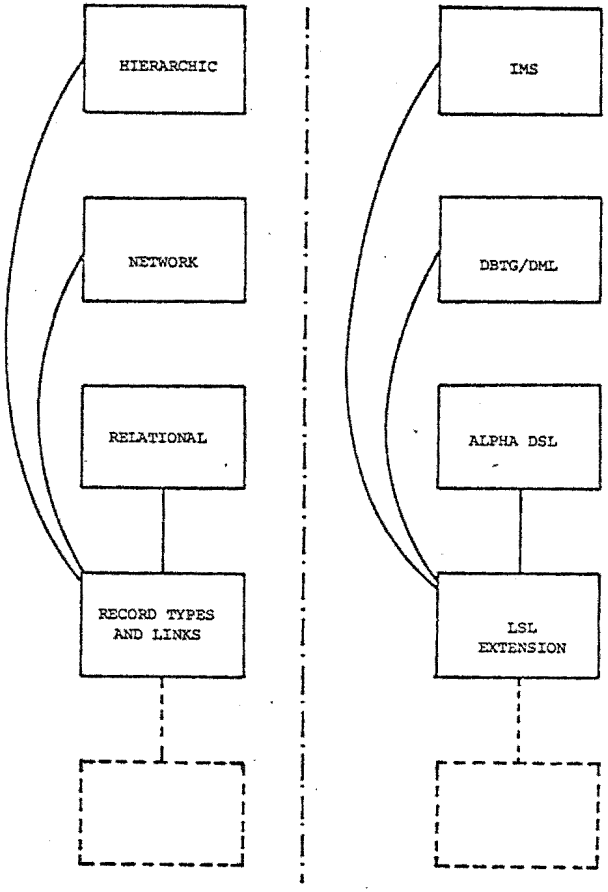


FIGURE 2.17 Examples of architectural representations

ZETA/TORUS:

A. INDEPENDENT MODEL SUPPORT

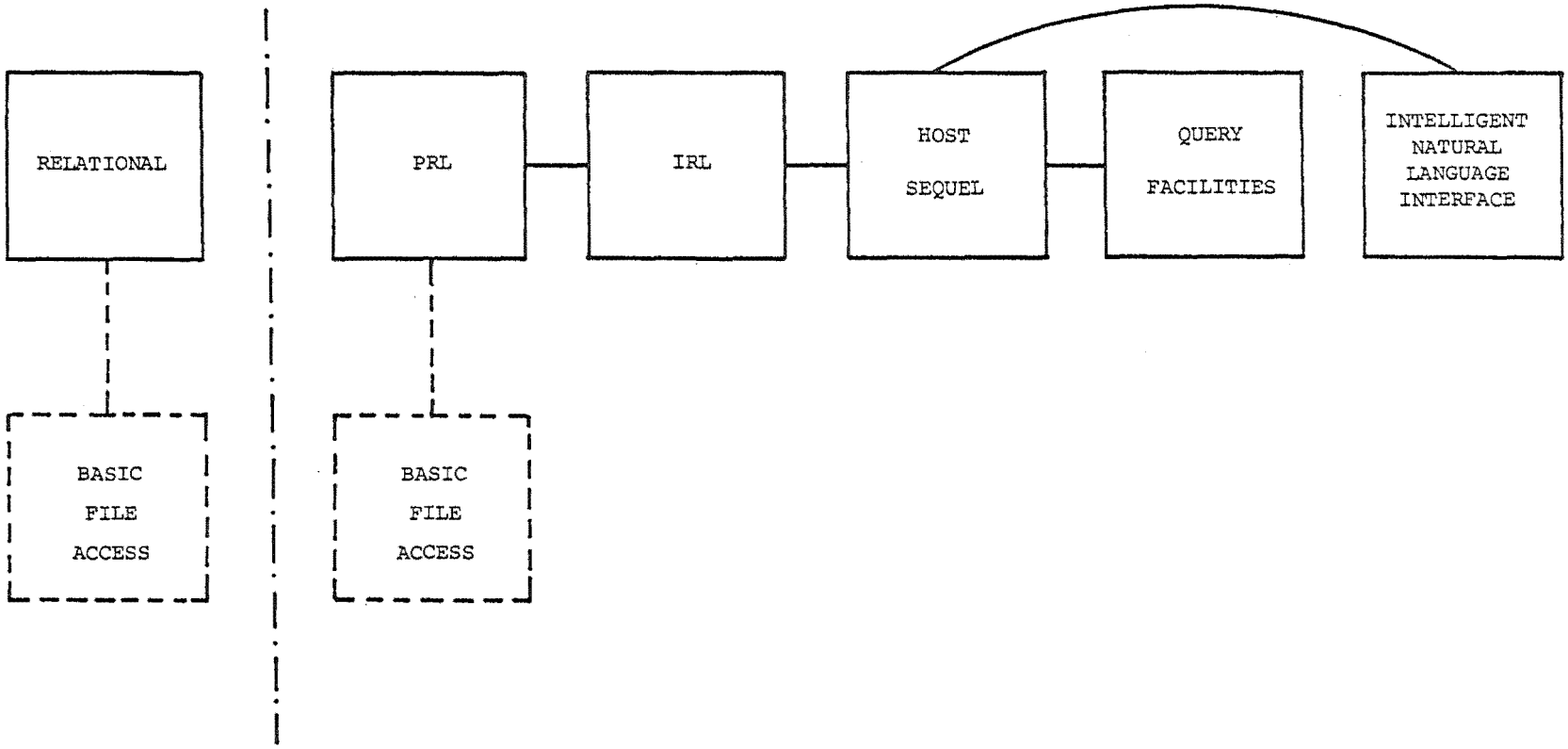


FIGURE 2.18 Alternative representation of ZETA/TORUS

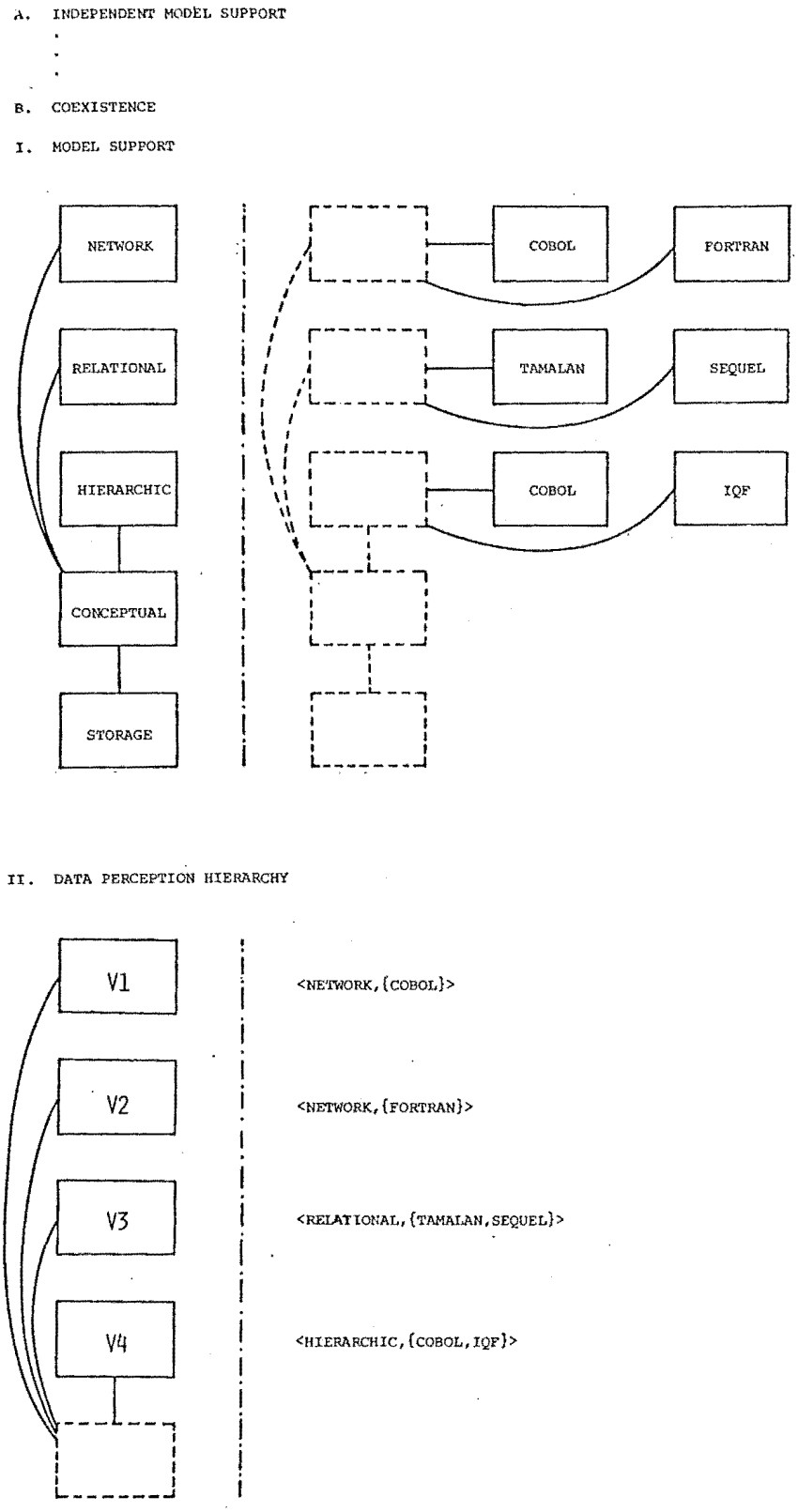


FIGURE 2.19 Alternative representation of Figure 2.14

CHAPTER 3

ABSTRACT MODELLING OF AN ENTERPRISE

3.1 INTRODUCTION

The *enterprise view* of data is the manner in which the enterprise perceives its data resource. The importance of a framework for the description of an enterprise view of data has lately been recognized (ANSI/X3/SPARC [1975], Chen [1976], Sibley and Kerschberg [1977]). Models for the representation of data (Chapter 2) fall into two types, abstract models and data models. They are distinguished by their approach in the modelling of an enterprise.

In the modelling of the real world, the concepts of the reality must be related to the objects of the data base. The relationship is specified in a stated perception of concepts of the reality. This stated perception or abstract model (Biller and Neuhold [1977]), provides the semantic reference with which one is to view the objects of the data base.

The network and relational models (Date [1975], Fry and Sibley [1976]), among others, have been proposed as representations for the logical view of data. One inherent weakness of these data models stems from the fact that they do not stress the enterprise view of data; that is, the abstract model or real world perception reference is not clearly defined. The approach of these data models has been to start from a representation and then define conditions and operations on the representation to reflect the dynamics of some specific enterprise concepts. In such a situation it is not clear what abstractions the model is meant to support.

An alternative approach is to create an abstract model by identifying and defining formalized data concepts and then proceeding to the representation. This results in data descriptions that are in much better accord with the enterprise view.

Various data constructs, many containing similarities, have been proposed and presented in diverse contexts and terminologies. This chapter represents a consolidation and extension of these concepts. A comprehensive set of abstractions of reality is described, based on a common framework. The approach presented here differs from those of conceptual frameworks found in the literature (e.g. Smith and Smith [1977a, 1977b], Biller and Neuhold [1978]). These approaches place emphasis on the form of fact representation, whereas this chapter stresses and puts into perspective the generalized constructs that are imposed on fact representations. Schmid [1977] proposes *complex object types* which are to provide the means for embedding rules as required in specific situations. The approach here goes further by isolating and defining generalized abstractions. These then form well-defined constructs to be used as templates in conceptual modelling.

Sec. 3.2 introduces the framework by describing the approach adopted here in the representation of the real world.

Ideas about the real world can be separated into two areas - one concerning concept types and the other concerning the instance values of these concept types. These two areas will be referred to here as the concept abstraction and value abstraction domains respectively. The two are not independent. For clarity however, the two areas are discussed separately. Sec. 3.3 describes concept abstractions while Sec. 3.4 discusses value abstractions.

Specific applications may not utilize all possible constructs. To cater for this, a spectrum is proposed in Sec. 3.5 which indicates grouping of abstractions providing a meaningful range of conceptual

sophistication in data base models.

3.2 REPRESENTATION OF THE REAL WORLD

The modelling of the real world can be seen to consist of two distinct parts. One part concerns the *static* representation of data which models the perceived information concerning the enterprise. This distinct level of fact representation has recently also been recognized by other authors - Falkenberg [1977] refers to it as the *deep-structure data model*, and it is also equivalent to choosing one particular representation of the *unstructured conceptual model* (Schmid [1977]).

The other part concerns the conceptual *dynamics* of the data reflecting the effects of real world operations. The dynamics of interest are those based not on the particular enterprise, but rather on the type of concepts modelled. The isolation of these concept-types involves an abstraction of reality, and it is these abstractions which constitute an abstract model.

Sec. 3.2.1 discusses the representation of facts, while Sec. 3.2.2 considers the representation of conceptual dynamics.

3.2.1 Representation of Facts

3.2.1.1 Predicate and object representations

A *fact* is the existence of a relationship involving definite objects, states and actions in the world that is being modelled (Wong and Mylopoulos [1977], Biller and Neuhold [1977]). A data base is the collection of data that represents those facts defined to be of interest to an enterprise (ANSI/X3/SPARC [1975]).

The input of this data may be of such a high level that they may be expressed to the data base as English-like sentences (Sibley and Kerschberg [1977], Mylopoulos *et al.* [1976]). However, the actual

interpretation of the data is determined by the form of the underlying conceptual structures into which the facts are assimilated and with which the facts are perceived.

Structuring in terms of individual facts presents a prohibitive complexity in the perception of what is contained in the data base and in the interrelation of the facts. The alternative, the construction of facts in terms of *types* can take one of two forms: as predicates or in terms of the objects involved (Schmid [1977]).

Both forms are equivalent in the sense that for a given set of facts, perceptions in both forms can be constructed to contain exactly those facts. This suggests the transformability of one form into the other. To illustrate the relationship between the two forms, consider the facts:

"supplier S1 supplies the part P1",

"employment agencies supply jobs", and

"suppliers supplies-to projects"

In the predicate form, the relationship types are strongly modelled. For example, in the first fact above, "supplies" is represented as the predicate

SUPPLY (S1, P1)

where the first item in the tuple represents the supplier, and the second the object supplied. Many-sorted logic (Wong and Mylopoulos [1977]) is used to allow the specification of the 'sort' of each item, e.g.

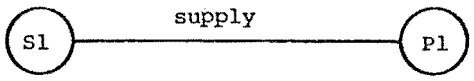
SUPPLY (SUPPLIER, PART) .

This is, in effect, specifying the object-types over which the predicate is defined.

The predicate model for the facts listed previously is given in

Fig. 3.1(a). The set of predicates do not emphasize the involvement of particular objects in different predicates, while the two applications of the predicate SUPPLY are immediately seen.

The opposite is true in an object-oriented representation. This representation involves the isolation of the objects of interest and the fact as associations between them. The representation is typically illustrated using graph forms, where nodes represent objects and lines their association. For example, the diagram for the first fact listed previously is



As in predicate representations, types of objects and associations are perceived, rather than instances. The object-oriented representation of the given facts is shown in Fig. 3.1(b).

3.2.1.2 Binary relational object representation

An object-oriented representation is adopted here since it is seen to be the more natural perception of the real world.

In this representation, the binary relational model is the most suitable form. Consider the fact

"Suppliers supply projects with parts on certain dates" -- (3.0)

This fact can be expressed by binary predicates (Wong and Mylopoulos [1977]) by introducing the *event* of supply, E:

ACT-OF (E, SUPPLY)
AGENT (E, SUPPLIER)
RECIPIENT (E, PROJECT)
OBJECT (E, PART)
TIME (E, DATE)

ACT-OF specifies E's type, while the other four predicates specify its arguments. This corresponds directly to an object-oriented representation of this fact illustrated in Fig. 3.2. The node E in

this figure represents the event E of the act of supply.

Each line corresponds to a binary predicate. Such a decomposition of facts into binary form is always possible (Levien and Maron [1967]) and is necessary in deriving the representation, since each line represents only a single association between two objects. This leads to the observation that binary relational models (e.g. Bracchi *et al.* [1976], Senko [1976b]) are the only purely object oriented representations, since any record oriented model would contain embedded predicates.

This feature of binary relational models, described variously as that of representing the most "basic facts" (Senko [1976b]) or the "smallest concepts" (Bracchi *et al.* [1976]) make them the most suitable candidate for object oriented fact representation. This representation is adopted here.

3.2.2 Representation of Conceptual Dynamics

The function of a fact representation model is to provide the mechanism to represent facts of interest to the enterprise.

However, this static representation of facts is insufficient. The data base is also required to model the dynamics of the real world. These take the form of constraints which are distinguished into two groups, one group consisting of constraints that are based on specific objects and the other those that can be generalized. The former can be embedded as *procedural semantics* (Sec. 3.3.10), while the latter group are those which form the basis of the abstractions in conceptual modelling. The rest of this section discusses these generalizable dynamics further.

These dynamics are seen as the consequence of differences in the perception of the objects and the implications of their roles in the associations. Of particular interest are those dynamics representing

conditions or constraints of *existence* and *identification*. The requirement for these constraints arises out of two considerations: first, the ability to view objects relative to their degree of interest, and secondly the limitations of space.

The first is generally acknowledged, although it is recently asserted by Smith and Smith [1978] that such relative interpretations should be relegated to alternative views and not to the data resource itself. However, the second consideration implies that only those objects of interest are to be maintained in the data base. That is, an agreed upon perception of relative interest also has to be imposed on the central resource to specify the dependencies of existence.

For instance, in the fact 3.0 given previously and its corresponding representation in Fig. 3.2, the object E, representing the act of supply, is the main concept of interest (Kerschberg *et al.* [1976], Sibley and Kerschberg [1977]) while the other objects are seen to merely describe it. This would create an existence relationship where the deletion of an instance of E would trigger the deletion of its descriptors in the other object-types.

These differences in the perception of object-types as well as those in the associations of the objects (Tsichritzis and Lochovsky [1976], Schmid and Swenson [1975]) have to be imposed on the homogeneous binary relational representation. This assignment of types determines the dynamics of the representation corresponding to the perception. These types of objects and associations perceived in the data base represent abstractions of reality.

The next section presents a comprehensive treatment of abstractions for the modelling of real world data concepts.

3.3 CONCEPT ABSTRACTIONS OF REALITY

3.3.1 Introduction

It is necessary to distinguish between the universe of real world objects and the universe of data base objects. The data base modelling of the real world presented here centers around the modelling of the existence and description of objects of the real world, and the associations between these objects. The description and relationships of objects in the real world vary in type and extent, with the state of the perceiver. However, data base objects have to be unequivocally described and have well defined associations at every instant. Abstractions serve this purpose by providing constructs with which the data base universe is to be defined. The formulation of different semantic constructs arises from the perception of different types of associations and descriptions of objects in the real world.

Abstractions are imposed on the binary model fact representation (Sec. 3.2) by assigning the objects and associations to particular classes. Diagrammatic conventions are introduced to differentiate classes of objects and associations in the binary model representation (e.g. Fig. 3.2). The main diagrammatic representations are given in Fig. 3.3. Diagrams of other concepts are introduced as they are discussed. To facilitate correspondence, these diagrams are similar to those used in other approaches (e.g. Chen [1976]).

In this thesis, the adjective *data base*, or the prefix db-, may be used to qualify an abstraction type (e.g. data base entity, db-attribute) to indicate that the abstraction type of the model is meant. When no qualification is stated, the implied reference to the data base or real world concepts will be clear from the context.

An abstraction is the formalization of a concept that can be applied, or be seen to exist in a large number of situations. Often, a concept may appear infrequently or its variations too numerous, or the concept is inherently specific to particular objects or its general characteristics are not clear. In such cases, an abstraction either cannot be constructed or is impractical to construct. The alternative is to provide the user with facilities with which he may embed these concepts into the system. Such a facility, with which to embed these *procedural semantics*, should be a part of any data base system, since it is unlikely that the set of abstractions provided by the system would cater for all concepts perceived in an enterprise.

The remainder of this section describes specific abstractions. Constructs found in the literature (examples of surveys and analyses are Kerschberg *et al.* [1976], Biller and Neuhold [1977], Wong and Mylopoulos [1977]) are encompassed.

Various extensions are introduced. In particular, the IS-A association (Wong and Mylopoulos [1977]) is seen as a special case of a more general IS-ONE-OF association, the PART-OF association (Smith and Smith [1978]) is described as a form of fact representation on which various abstractions can be imposed, and an interpretation is given for the specification of cardinalities of involvement in db-events.

Sec. 3.3.10 classifies concepts representable by some embedded procedural semantics.

The following data base abstractions and concepts are identified:

A. Generalized constructs

A1. Basic concepts

1. *Entity* which models the existence of a real world object and its capability of being described.
2. *Attribute* which represents the concept of the description of an object where the descriptor object is not itself described.
3. *Relationship* which represents the concept of the mutual description or association between two entity or event types.
4. *Event* which models a describable object in which the object represents an association between entity or event types.
5. *Dependent object* which represents an object whose existence is determined by another object through an association.

A2. Associational concepts

6. *Cardinality of mapping* which represents the cardinality of an object in its role in an association. Two levels are distinguished at which this concept can be modelled.
7. *IS-A association* which represents the associations of conceptual containment among entities and among attributes. An extension of this is the

   IS-ONE-OF association which allows an object set to have members which are also members of one of a number of other entity sets.
8. *Composite attribute* which represents the associations of composition among attributes. It is seen as a construct imposed on a particular form of the PART-OF representation of facts.

B. Concepts through procedural semantics

1. *Membership associations* where the association between objects of specific sets are defined with respect to the cardinality of the association or on the value of the objects.
2. *Derived objects* which represent the construction of objects from other objects through some procedure.
3. *Associational semantics* which represents any interaction based on the meanings and interrelations of associations.
4. *Semantics of time* which includes any concept concerning the objects and associations in which there is an involvement of time.

Fig. 3.4 describes an enterprise situation to be used in illustration. It lists the information that is to be represented. The binary relational model corresponding to this list is given in Fig. 3.5, where all the objects and associations of interest are represented. The process of abstraction on this consists of determining the constructs which best reflect the role of particular objects in the perception. This involves specifying the degree of interest of each object. That is, it is based on the perception of the enterprise at a particular time.

3.3.2 Entity

The most basic abstraction is that of the *entity* which represents a real world entity. ANSI/X3/SPARC [1975] defines a real world entity as "a person, place, thing, concept or event, real or abstract, of interest to the enterprise". However, since all objects to be represented are necessarily of interest, the definition above is qualified here to say that a real world entity is an object of *main* interest to the enterprise (Kerschberg *et al.* [1976]). It is also

uniquely identifiable, independently existing and perceived to be described by other objects. Db-entities, being representations of real world entities, therefore also have these properties.

A real world entity set is a collection of real world entities of the same type. This is reflected in the data base by a data base entity set which is a collection of data base entity instances. For each perceived entity-type there corresponds an entity set.

The choice of which object-types are to be assigned as entities is determined by the perception of the enterprise. Fig. 3.6 illustrates the abstract model of the enterprise. In the model of Fig. 3.5, the object-types perceived to be entities are indicated by the rectangles in Fig. 3.6. The entity, Vehicle, is seen in this perception to exist independently of the other objects.

3.3.3 Attribute

Associated with an entity or any describable object may be several *attributes*, representing the descriptions (properties, characteristics) of the real world object which the described data base object is representing. In the real world an object perceived as a property (of another object) may also be viewed as an entity in its own right. An example is *colour* which may be used to describe an entity, say *car*; but *colour* itself may be an entity in that it may be described by, for example, *wavelength*. This multi-context use of the same object is always possible in the real world. However, the contexts themselves are well-defined. An object is a property when it is being used to describe another object, it is an entity when it itself is being described. In data base abstractions, a db-entity is described by db-attributes or, equivalently, db-attributes describe a db-entity. That is, the role of a data base object as attribute or entity is fixed.

Those object-types of Fig. 3.5 modelled as attributes are shown as circles in Fig. 3.6. The associations of an attribute to the objects it describes are indicated by the dotted lines. An attribute cannot exist on its own, since it is meaningful only as the description of some object. It may however be *defined*, which is distinct from existence.

3.3.4 Relationship

In the real world, situations often arise where an object is perceived both as described as well as being a descriptor. The *relationship* abstraction type reflects this associational concept. The description of an entity by another entity is differentiated from the description by an attribute (Schmid and Swenson [1975], Tsichritzis and Lochovsky [1976]). The former is called a data base relationship and the latter an *attribute association.* Both associations model the *concept* of describing. They are description types and correspond (in English) to verb phrases - e.g. "owns car", "can be contacted by phone number". However, a relationship facilitates an association between two entities, while in an attribute association the descriptor object is not itself described.

In Fig. 3.6, the entities Employee and Project are joined by a double-line, indicating their mutual description through a relationship. There may be many relationships between any two entities, that is, the entities may describe each other in many ways. Fig. 3.6 shows two relationships between Supplier, S and City, C. The relationship, R2, can be viewed as "Suppliers S have major locations in Cities C" or the inverse "Cities C are the major locations for Suppliers S". The other relationship, R3, between S and C indicates "auxiliary location".

3.3.5 Event

The *event* abstraction type represents the real world viewing of a descriptor type as a describable object. For example, the description

type "owns car" (person : car association) may be viewed as the (event) object "ownership of car" which might be described by, e.g., "date of ownership" and "conditions of ownership". The event "ownership of car" is the noun-form of the verb phrase "owns car" (Cadiou [1976]).

In Fig. 3.5 the substructure, X, represents the two facts (3) and (4) of Fig. 3.4. The correspondence can be seen by decomposing the facts into a nested binary relational form (Levien and Maron [1967]):

A: "suppliers supply projects"
B: "parts are the object-supplied in A"
C: "B is in some quantity"
D: "A is described by its regularity"

That is, the object E1 is introduced to represent the association A since it participates or is described in the association B. Similarly, the association B which is described in C is represented by the object E2. In other words, E1 and E2 are events representing binary associations which are viewed as describable objects.

This is indicated in Fig. 3.6 by the two diamonds E1 and E2. The event E1 represents the act of supply of Suppliers to Projects. The entities Project and Supplier are said to be *involved-in* the event E1. This association is indicated by the solid lines. E1 is described by regularity of supply (which is modelled by an attribute) and by the object supplied, Part. However, since this association of Supply with Part is to be described by its Quantity, the act of Supply of Part is modelled as the event, E2. As Part is modelled as an attribute, E2 represents an event of the attribute association of an event, E1, and Part. Where it is not clear (as in the involvement of E1 in E2), an arrow is used in the diagrammatic representation (Fig. 3.6) to indicate the direction of involvement.

The perception of a describable event implies cognizance of the objects and the association from which the event is derived. It therefore cannot exist independently. Also, the specification of a data base event contains the specification of its corresponding associations. Their separate specification would result in conceptual

redundancy as well as problems in specifying their equivalence. So for example, the event of supply involving Supplier and Project contains the relationship of supply between these two objects.

The distinction between relationships and events explains to a certain extent the view that *excess entities* (Senko [1976b], Hall *et al.* [1976]) do not exist in this general framework and, by implication, in binary relational models. An excess entity is defined as an object which is created by necessity of the model rather than of conceptual interest. It is typically associated with the creation of default objects in the representation of relationships, for example as in Chen's [1976] ER-model. In the framework presented here, however, the association object or db-event, is created only when the association or db-relationship is in fact of interest as an object. In this sense the object created has a corresponding real world concept and cannot be viewed as an excess object.

3.3.6 Dependent Object

A dependent object is one whose existence is determined by the existence of another object. An attribute is a dependent object since it exists only as a description of some other object. Situations also arise however, where a describable object or entity is perceived as being dependent. Such an object is called a *dependent-entity*.

For example, in the corporate data base of Fig. 3.5, the object Dependent-of-Employee, which is described by Date-of-Birth and Name, may be perceived only in the context of the Employee entity on which it is dependent (Chen [1976]). Thus the deletion of an Employee from the data base requires the deletion of all the Employee's dependents as well as their attributes. Also, the identification of a Dependent requires, and is to be only possible through, the identification of the related Employee. In terms of processing this means that the processing of

Dependents is only to be done within the processing of Employees. Fig. 3.6 illustrates this situation, where the dependent-entity, Dependent-of-Employee, is drawn with dashed lines. The relationship, R_d, between the Dependent-of-Employee entity and Employee entity (through which the former is dependent on the latter) is called a *dependency-relationship*. If this dependency-relationship is itself described, then the resultant event is a *dependency-event*. Fig. 3.7 shows the dependency-event, E_d, obtained when the dependency-relationship, R_d, of Fig. 3.6 is perceived as a describable object.

A dependent object may also be a component of a regular event. For example, in Fig. 3.7 the dependent entity, Dependent-of-Employee, together with the entity, Company-Car, form the regular event, Usage-of-Company-Cars.

3.3.7 Cardinality of Association

An association between two object-types represents a specification of the association of instances of the two object-types. It is of interest to be able to specify the cardinality of the association between instances of each object-type. There are two levels at which this concept could be modelled. They are, in increasing order of sophistication, the *(generic) cardinality of mapping*, and the *interval (cardinality) of mapping*.

Of these, the cardinality of mapping is the most widely referred to. It describes the cardinality of the associations between unique instances of object-types in terms of the mathematical mapping types of one-one (*1:1*), one-many (*1:n*) or many-many (*n:m*). An example is given in Fig. 3.8. The *1:1* association between Project (P) and Schedule (SC) (denoted Project $^{1}:^{1}$ Schedule) indicates that any single P can have at most one SC and vice versa. The Project $^{n}:^{1}$ Special-Equipment (SE) association means that any single P may be associated with at most one

SE, whereas any particular SE may be associated with an arbitrary number of different P. Similarly, the Employee (E) $^{n}:^{m}$ Project association indicates that an E may be associated with many P and vice versa.

The interval of mapping provides for the specification of the explicit cardinality of an association. Each object-type in an association is said to play a particular *role* (Falkenberg [1976], Bachman and Daya [1977]) in the association. The interval of mapping indicates explicitly the cardinality of each object in its role (Abrial [1974]). For instance, the role of Employee in the relationship, R1, of Fig. 3.6 is "project member", while the role of Project may be described as "the project to which members are assigned". The specification Employee $^{\langle 0,3\rangle}:^{\langle 4,8\rangle}$ Project denotes that an Employee does not necessarily have to be a member of any project although he is permitted to be a member of at most three different projects. The specification also indicates that each Project is to have between 4 and 8 members assigned to it.

This can be extended to describe the cardinality of roles with respect to events. Consider the Supply of Parts by Suppliers to Projects represented in Fig. 3.6. The Project : Supplier association is known to be *n:m*. However, it may also be required to indicate the cardinality of the involvement of Projects, Suppliers and Parts in each act of or event of Supply. An example is given in Fig. 3.9. Each event of Supply would correspond in the real world to a contract incorporating conditions on the involvement of Projects, Suppliers and Parts in the contract.

The Project $^{\langle 1,\infty\rangle}:^{\langle 2,2\rangle}$ Supply association indicates that each Project is involved in at least one contract of Supply, but that any single contract must involve exactly two Projects. That is, contracts can only be drawn for pairs of Projects. Similarly, the Supplier

$\langle 1,\infty \rangle : \langle 1,3 \rangle$ Supply and Part $\langle 1,\infty \rangle : \langle 1,4 \rangle$ Supply associations specify that each contract can involve between 1 and 3 Suppliers, which together can supply between 1 and 4 different Parts for that contract. Both Suppliers and Parts can be involved in many contracts. A corresponding interpretation for cardinalities of mapping of events can also be made.

If known, specifications of cardinality or interval of mapping can be included in representations such as that in Fig. 3.6.

3.3.8 IS-A Association

In the concept representation of Fig. 3.6, the object-types are viewed as being distinct so that the corresponding sets of object instances are presumed to be disjoint. To model situations where concepts are not distinct, further specifications need to be imposed on the objects to indicate their associations of membership. These associations can be diagrammatically illustrated separately from those in the representation of facts.

Associations of membership between sets of objects may follow arbitrarily complex rules. However, it is the function of abstraction to isolate and characterize specific associations of interest. One particular association of membership concerns that of total containment of object types and therefore object sets. This association is called the IS-A association (Wong and Mylopoulos [1977]).

An example is given in Fig. 3.10. The arrows indicate the direction of containment. For instance, the specification Manager IS-A Employee means that at all times each Manager must also be an Employee. In other words, Manager is a *sub-entity-type* of the entity-type Employee. Similarly, the attribute Car-Body-Colour (CB) is a sub-attribute of Colour-used-in-Car (CU), meaning that any descriptor value available as CB must be one of those available as CU.

IS-A associations can only be defined between objects of the

same concept-type, since for example, it is contradictory to view an entity to be also an attribute, or vice versa. Often, associations of containment form hierarchies, as that in Fig. 3.10(b). Objects which are contained by the same object may *overlap*. However, this may not always be so, since, for example, Car-Interior-Colour may or may not have common values with Car-Body-Colour. Recently (Lee and Gerritsen [1978], Hammer and McLeod [1978]), various forms of overlapping have been classified and it is seen that the rules of intersection are typically specific to the situation. These can be included as a form of procedural semantics (Sec. 3.3.10).

Associations of containment may not always result in strict hierarchies. In Fig. 3.10(a), Customer is a Person, but also Customer is a Corporation. Such a situation occurs since there may be a many-many association between roles and entities (Bachman and Daya [1977]) and that overlapping entities arises from the separate perception of each set of entities in a particular role. This situation, where members of an object set may come from several other object sets is more general than that modelled by the IS-A association. This more general association is called here the IS-ONE-OF association. The IS-A association is viewed as a special case in which an object-type IS-ONE-OF exactly one object-type.

3.3.9 Composite Attribute

The PART-OF association (Wong and Mylopoulos [1977]) is used to specify the components of a concept or to group several concepts into a unit so that they can be treated as a whole. This association has been used by Smith and Smith [1977b] as the basis for modelling the real world. However, it is essentially a methodology to discern facts of the real world. Conditions of existence still need to be imposed, as indicated by a further paper of Smith and Smith [1978] in which the

concept of 'principle of individual preservation' is introduced and by Navathe and Schkolnick [1978] who includes 'instance-level interrelationships among data' into the aggregation/generalisation model.

In a generalized PART-OF modelling of the real world, the objects and PART-OF associations constructed can be imposed upon by some abstraction. This is discussed below where composite attributes are seen to correspond to a particular form of PART-OF association.

Consider for example the *aggregation* (using the terminology of Smith and Smith [1977b]) in Fig. 3.11, similar to that given by Smith and Smith [1977b]. The assignment of abstractions on this model involves forming a judgement of the role of the objects and their interrelationships. A particular interpretation based on the description in Smith and Smith [1977b] is given in Fig. 3.12. As a guide, the lowest level primitive objects are normally seen as only the descriptors (i.e. attributes) of higher level objects, while the higher level objects are usually modelled as entities or events. However, as can be seen from Fig. 3.12, this is not always so. Keys, e.g I-Number, correspond to tokens (Sec. 3.4.1) and represent the object (Instructor) itself. To indicate the correspondence, token value ranges (Sec. 3.4.1) are shown with the same name as the aggregation keys.

From this figure it can also be seen that except for the associations among the attributes, all the PART-OF associations of Fig. 3.11 are mapped to some association of the abstractions described in the previous subsections. This association of composition among attributes represents a further abstraction, which in principle is commonly available (cf. COBOL groups of elementary fields).

An attribute that has other attributes as components is called a *composite attribute*. For example, in the substructure Y of Fig. 3.12, the composite attribute S-Birthdate is composed of the attributes

S-Birthday, S-Birthmonth and S-Birthyear. The manifestation of this concept is the availability of a composite attribute instance as the physical concatenation of the instances of its components. Therefore an ordering (left to right in Fig. 3.12) of the components is required to indicate their order in the physical concatenation.

As with IS-A associations, the composition of attributes can also be indicated separately from the concept representation of Fig. 3.6.

3.3.10 Procedural Semantics

Facilities are required to express those constraints specific to particular objects and not catered for by generalized abstractions. For example, consider the following conditions to be incorporated into the model of Fig. 3.6.

(1) "every supplier supplies at least one project with all parts" and

(2) "every supplier who supplies part P1 has its major location in city C1".

The abstractions of Secs 3.3.2 to 3.3.9 cannot explicitly represent these constraints.

Owing to the immense range and complexity of such rules, it is difficult to construct generalised abstractions for these associations. Instead, the approach is to represent the perceived semantics as procedures inserted between the user and the system. These *procedural semantics*, which describe the rules of consistency to be maintained by the system, are variously referred to as *integrity, consistency* or *semantic constraints*. It can be expressed in many forms, in particular as *assertions* (Chamberlin [1976]) in terms of the data base language provided by the system. For example, it may be specified using QUEL expressions (Held *et al.* [1975]) or SEQUEL expressions (Chamberlin *et al.* [1976]), in a procedural form (Machgeels [1976]) or as more

formalized predicate calculus expressions (Wong and Mylopoulos [1977]).

In data models, where no abstractions are provided, all the rules have to be explicitly entered in terms of procedural semantics. The differentiation of these constraints into classes (e.g. Benci *et al.* [1976], Machgeels [1976], Weber [1976]) is the data model substitute for the abstractions in abstract models.

In abstract models, any concept in the modelling of the real world not representable with the abstractions of Secs 3.3.2 to 3.3.9 is relegated to procedural semantics. Although it is not possible to enumerate all concepts, it is useful for further development to characterize groups of related concepts. The following are brief characterizations of four classes of concepts.

3.3.10.1 Membership assertions

These assertions specify the rules under which objects or associations are to exist, based on

(1) quantifiers, with respect to related objects or associations, and/or

(2) the values of related objects.

An example of each form is given in the beginning of this section. Another example is the constraint that "for any given election the number of votes received by the winner is greater than the number of votes received by any loser" (Chamberlin [1976]). This assertion could be of either form (1) or (2) depending on whether the votes are recorded individually or as the total received. Note that while the interval of mapping (Sec. 3.3.7) specifies only the range of the associational cardinality between the objects of two sets, here the specific cardinality of individual objects over possibly many associations may be involved.

3.3.10.2 Construction of derived objects

This specifies the construction of the value of an object from other objects by some computation. The value of the target object would have to be updated whenever any of the objects on which it is defined is updated.

Immediate examples are any statistical attribute involving a few object-types, e.g. "number of different parts", "average grades" or "ratio of male/female employees". However, it is possible, at least in concept, to have derived objects whose arguments involve an arbitrarily large range of objects and conditions; for example: "the number of secretary/manager pairs, who have worked together at least 3 years, have average age less than 35 years, and where the manager's salary >$16,000, etc.". This is, in effect, representing concepts typically obtained with an application program as an object of the enterprise model, by embedding the program into the system.

3.3.10.3 Associational semantics

In the abstractions considered thus far, the emphasis has been on the objects and not on types of associations. Associational semantics are those involving manipulation based on the meaning of specific associations or roles in associations.

As examples of these:

(1) Consider the situation where Persons may have relationships of "owning" with several object-types. Then a Person's destitution which entails his relinquishing all that he owns, would require the access of all the owning relationships which may not all be called by the same name.

(2) Consider the model in Fig. 3.13(a). The relationships between Persons model different aspects of close family ties. For the query "do any of Person X's *relatives* earn more than $16,000 a year?" to be answered by the system requires the system knowing

that *relatives* are available through these various associations. Note that even though sub-entities can be constructed for the Persons in a particular role, as in Fig. 3.13(b), generally the relationships still need to be specified, since there may be other relationships between the entities. For example, in Fig. 3.13(b), of the two relationships between Person and Parent only one is guaranteed to obtain a particular Person's *relative*.

These examples suggest that generalized abstractions can be formulated to specify conceptual containment among associations. In most cases, however, the support of constraints based on the meaning of associations would require some form of procedural semantics.

3.3.10.4 Time dependence

The role of time in a data base system can take two forms:

to *action triggers*
or to *maintain history*

The manipulation of objects and associations involving time is in terms of one of these roles. As in the other dependency classes, arbitrarily complex time dependencies can be perceived in the real world.

Some relatively simple examples of practical interest in the use of time as a trigger are:

(1) "put status = overdue if supply not received 2 weeks after order",

(2) "a child can no longer be a dependant when the child reaches the age of 18 years".

Versions (Grotenhuis and van den Broek [1976]) of data can be kept to enable queries, manipulation, or even objects of the model to be based on history. Examples based on the past are:

(1) "if patient has had strokes in the last 6 months, mark 'special consideration'".

(2) "what is the mean annual population growth rate in City C1

between the years 1960-1965?"

Physical limitations means that the recording of versions themselves may be subject to time constraints, for example:

(1) "keep addresses of employees to 3 years back",

(2) "retain the car-reg# of all the cars a person has ever had".

These, however, can be viewed as further examples of triggering.

3.3.10.5 Discussion

Although the four forms of procedural semantics described in the previous subsections are individually characterized, any particular condition may involve all the concepts. The characterization, however, enables a systematic approach in providing support for these concepts. Of the four areas described, the first two: membership assertions and derived objects, are the concepts most widely referred to. Approaches in integrity constraints are essentially facilities to support specific constraints in these two areas. Associational semantics have not been treated rigorously in data base management, but are considered in the field of artificial intelligence (Wong and Mylopoulos [1977]).

Study of the representation of time is still in its infancy, but is rapidly gaining in significance (Sundgren [1975], Benci *et al.* [1976], Bubenko [1977], Senko [1977b]). Bubenko [1977] presents a conceptual framework which includes the 'temporal dimension'. Bubenko illustrates, however, that most data base models can incorporate time by introducing a special object to represent it. Senko [1977b] outlines the application of this in DIAM II. A similar approach can be taken within the framework of this chapter. In any case, systems which incorporate time considerations require each fact entered to be accompanied by a time parameter.

3.4 VALUE ABSTRACTION

3.4.1 The Instances of Abstractions

The modelling of the real world is the specification or the fitting of real world concepts as abstraction types using the available abstractions (as defined, for example, in Sec. 3.3). The defined abstraction types and the associations among them represent the agreed upon model of the real world concept types of interest to the enterprise. The *instances* of these real world concepts are represented by the value instances of the abstraction types and their associations. Concepts for the representation of value abstractions are presented in this section. These include the instance set, range set, tokens and identifiers. The diagrammatic representations for these concepts are given in Fig. 3.3(b).

The representation of dependent objects (entities and events) is discussed in Sec. 3.4.2. The manifestations of the existence dependencies of other concept types as instance dependencies are discussed in Sec. 3.4.3.

3.4.1.1 Instance set

The association between the data base objects is represented finally by the association between values for these objects. Thus an entity-attribute association, for example, is represented by the association of values from two sets - one representing the instances of the entity type and the other the set of attribute-instances. For each abstraction type, then, there exists a set of values which participate in the association instances to represent the values the real world is perceived to have at that given instant of time. The sets viewed in this form will be called *instance-sets*. Within each instance set any identical values pertain to the same object. This is not necessarily so for different instance sets. For example, the

person BROWN and the colour BROWN have the same value, but are of different ranges (described below) and different concepts.

3.4.1.2 Range set

Each unique value from the instance sets for an abstraction type is a member of a set of values. This set represents the allowable values that the corresponding real world concept may have at any time. Such a set will be called a (value) *range set*. These range sets are therefore the abstractions of the instances that real world concepts may have. Fig. 3.14 shows the range sets for the attributes of Home-address and Office-address. These attributes describe the entity, Person.

Range sets, as with sets in general, may have arbitrary membership. For example, a range set may consist of specific items (e.g. hair colour) or an interval (e.g. person weight).

3.4.1.3 Token

The instance sets corresponding to each of the functionally different concept abstractions (that of being described and that of describing) are treated separately because of the different semantic interpretations. Identical values in instance sets of entities and events represent the *same real world object*, while identical values in instance sets of attributes represent the *same description* of different objects. In the instance sets of describable db-objects each unique value represents a unique object in the real world. These values will be called *tokens*.

In Fig. 3.14 the range set for the entity Person is a token range set as each value represents a unique describable object. As events are describable objects, they also have token range sets in the value abstraction domain. Thus, for example, each instance of the events E1 and E2 in Fig. 3.6 has a corresponding token.

3.4.1.4 Identifiers

Often it is possible to identify a unique object in terms of its descriptions. For example, a Person may be identified uniquely by the Person's Name, Weight and Height if it is perceived that no two Persons have the same values for all of these descriptors. Such a combination of values of associated attribute instance sets used to identify an entity (or event) is termed an *identifier*. This is illustrated in Fig. 3.14 where the identifier set {Name, Weight, Height} is considered equivalent to the token set, I.D.#, and this equivalence is indicated by the dotted line.

The concept of identifiers is different from that of tokens, since an identifier merely identifies an object while a token is the object. To illustrate this, consider the example in Fig. 3.14 where the entity-type Person, with token range-set I.D.#, has the attribute, Home-address, and it is perceived that Home-address is an identifier. Consider further that the Home-address of a Person with I.D., #x, say, is deleted (representing, for example, that the person has moved but has not yet found a residence). Then although it is still true that a Home-address, where it exists, uniquely identifies a Person, it is now not possible to identify Person #x through Home-address, since this Person's Home-address does not exist. The existence of the Person in the data base is reflected by the token, I.D.#, and not by Home-address. If the actual person leaves the perception of the data base, then its token is deleted together with all its attribute associations. To further illustrate the difference in concept, it may be that at a later time it is to be perceived that different Persons may have the same Home-address. In this case, Home-address is no longer an identifier of the entity type Person.

3.4.2 The Representation of Dependent Objects

It has been stated that the existence of a describable object is represented by a token, and the object may sometimes also be identified by its attributes. A token range set can always be defined for any describable object type (entity or event). Situations arise, however, where the existence of an object is determined by the existence of another object.

For example, in Fig. 3.6, Dependent-of-Employee is perceived only in the context of the Employee entity on which it is dependent. This entity has no tokens, as implied by the dependent identification, since the existence of tokens allows direct identification of the object. Instead, the identifier for this dependent entity is a concatenation of the token of the entity (Employee) on which it is dependent together with appropriate attributes of the dependent-entity itself. Fig. 3.7 shows the identifier for the Dependent-of-Employee entity being made up from the token for the Employee entity and the dependent's Name attribute.

A *dependent entity* is therefore defined as an entity which does not have a token and its identifier contains the token of the entity on which it is dependent through a relationship. The owning entity may itself be a dependent entity. In this case the identifiers of all the nested dependent entities relate to the token of the final owning entity.

Any db-event that has a dependent object as a member does not have a token. Its identifier consists of the identifier of the dependent object and the tokens of the other member objects. For example, the identifier of the dependency-event E_d in Fig. 3.7 consists of the token of the owning entity (Employee) and the identifier of the dependent entity (Dependent-of-Employee). Since the token of the owning entity is already part of the identifier of the dependent entity, the

identifier of the dependency-event therefore has the same range set as that of the dependent-entity (although their instance sets are necessarily different).

3.4.3 Instance Dependencies

Although existence constraints (in the form of abstractions) are specified on object types, it is understood that they also apply to instances. This section describes particular manifestations of these dependencies.

3.4.3.1 Attribute-object dependencies

The instance dependencies of attributes to entities and events, as well as those between events and entities, have been described by Chen [1976]. One aspect that has not been treated is the differences caused by varying approaches in attribute perceptions. In particular a distinction is made between *factored attributes* (Sec. 4.2.2) where attribute instances are not shared among entities, and the general case where it may be shared. In the former the existence of an attribute instance is determined by only one object, while in the latter by possibly many objects.

3.4.3.2 Composite attribute instances

Composite attributes are those which have other attributes as components. An instance of a composite attribute is the concatenation of the instances of its component attributes if they exist. The deletion of a composite attribute instance therefore means the deletion of its component instances. It is possible to delete any component instance independently of its composite instance.

3.4.3.3. IS-A instances

IS-A associations, representing associations of containment between objects can be specified at various levels. In the general case, it could specify containment in terms of any of the value

abstraction concepts (value, ranges, instances) associated with the objects.

The particular specification for an object determines the dependency rules that are to prevail. Typically, however, IS-A associations among describable objects pertain to containment of instance sets, while those among attributes pertain to containment of range sets. That is, IS-A among attributes are based on permissable values, while among entities it is based on existing values.

3.4.3.3.1 Sub-object dependencies

Consider sub-entity types which represent conceptual subsets of another entity, called the source entity. For each token of the sub-entity there is a corresponding token of the source entity such that they both represent the same real world object. Thus, any operation on the token of a sub-entity is applicable to that token in the source entity, while for the reverse operation, this is true only for those tokens that also exist for the sub-entity.

To illustrate this, consider the example in Fig. 3.14, where Father is a sub-entity of Person and the containment of the Father token instance set in the Person token instance set is indicated by the arc in the value abstraction domain. The perception of a new Father, indicated by the creation of a new unique instance for the Father sub-entity type, would trigger the entering of this new instance into the instance set of the Person entity (if it is not already there). The reverse case, however, is not immediate. To include a new Person instance into the Father instance set requires that the Person be male and has children. This condition, however, cannot be established if sex and children are not modelled as attributes of Person. Thus, an external decision is required in such a case. Recently, Hammer and McLeod [1978], have also made this distinction between system controlled inclusion criteria and those controlled externally.

The deletion of sub-entities can be considered from two different aspects. In one, the deletion is seen as the exit of the object from the data base. This *global* operation would require the deletion of all tokens of that object including that in the source entity. The *local deletion* of a sub-entity, however, represents the exit of an instance only from a particular sub-entity type. For example, in Fig. 3.10(a), Trucker is a sub-entity of Employee. If a particular Trucker changes his occupation within the company to, say, a clerical worker, then a local deletion is used to remove the appropriate token from the Trucker sub-entity set. That is, in a local deletion, corresponding tokens in the source entity set are not affected. In contrast, the deletion of a source entity necessarily results in the deletion of the corresponding tokens in any of its sub-entity sets.

A similar process can be described for the sub-objects of db-events.

3.4.3.3.2 Sub-attribute dependencies

For attributes, a global deletion is an operation on the value ranges rather than on instances. For example, in Fig. 3.10(b), the deletion of RED, say, from the attribute Colour-Used-in-Car (CU) means that RED is no longer available as a descriptor in any of CU, CB and CI, rather than the simultaneous deletion of actual descriptions of objects. This allows local deletions of source attribute instances to represent their non-use as descriptors.

3.5 CONCEPTUAL SPECTRUM OF DATA BASE MODELS

3.5.1 Introduction

A *conceptual spectrum* of data base models is given in Fig. 3.15. Each point along the spectrum represents an increase in modelling power by the inclusion of further constructs. The spectrum illustrates a

progression of conceptual complexity which represents classes of systems which may be suitable to particular applications. Ideally, systems should be able to evolve easily from one point to another, since the spectrum corresponds to a natural progression in the information modelling requirements of a growing enterprise.

The following Subsecs 3.5.2 to 3.5.6 describe the characteristics of each of the classes. The discussion clarifies the role of particular concepts in the modelling of an enterprise. The relationship of the spectrum to existing approaches is given in Subsec. 3.5.7.

3.5.2 Primitive Systems

A system which models only entities and attributes is a *primitive system*. No association can be made among types of entities or attributes. Any record-handling system, including those prior to the advent of generalized data base systems, are effectively, such a system. Each record represents an entity and all its descriptions, so that a file represents a set of entities and its attribute associations. The concept of a key corresponds approximately to that of a token, where the deletion of a key requires the whole record to be deleted. In the relational model, if no association is permitted between tables (viz. no joins or other interrelational operators) then it would be a primitive model.

In applications where only information concerning isolated groups of objects are to be considered, such a system would be sufficient. Such applications would typically be either small and personalized or large involving simplistic mass processing of data. The minimal concepts provided, would however, make it unsuitable for most applications.

3.5.3 Basic Systems

A primitive system to which is added the capability of defining

relationships between entities, is called a *basic system*. Two forms of associations are now available: attribute associations, and associations between describable objects. It is desirable, therefore, to include the capability to specify cardinality of mappings in basic systems.

In a file-oriented system, the inclusion of relationships means that associations can be specified between records of different files. Consider the two record-types

PERSON (NAME, CAR)
CAR-INFO (CAR, COLOUR)

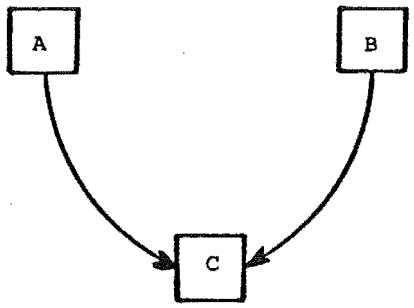
where PERSON represents the association of persons with cars, while CAR-INFO contains descriptions of the cars. In primitive systems no association is made within the system between the field CAR in PERSON and that in CAR-INFO. In a basic system the two uses of CAR can be specified to represent the same thing. This is essentially the effect of the join operator of the relational model, while in the network model a specific *1:n* relationship (set-type) construct is provided.

The bulk of existing data processing installations employs basically the three constructs of a basic system. This, however, is attributed to the constructs provided by commercial systems and not necessarily to the adequacy of the three concepts. In particular, a limitation of a basic system is the inability to view an association as an object in its own right.

3.5.4 Intermediate Systems

Many applications require the perception of an association between two objects as a describable object. For example, the act of a student taking a course is typically described by a grade. An *intermediate system* incorporates the db-event construct to model such concepts, as well as the constructs of a basic system. This set of constructs provides a sufficiently powerful modelling tool in the majority

of applications. The usefulness of events is indicated in network models by the frequent use of the ambiguous structure (Taylor and Frank [1976]) given below, where a record-type owned by two others is used to represent the event of their association.



3.5.5 Evolutionary Systems

An *evolutionary system* is an intermediate system with the addition of the concepts of dependent objects, IS-A associations and composite attributes. With these concepts, almost all of the generalized abstractions described in Sec. 3.3 are available. This means that detailed object perceptions of the enterprise can be modelled by the system. However, such a detailed appraisal of the enterprise is subject to frequent re-evaluation.

In primitive systems, changes in the perception of objects from attributes to entities and vice versa are possible. Perceptions with these concepts, however, are relatively more stable than those involving relationships and events in an intermediate system. For example, the change of perception of the relationship "owns-car" to the event "ownership-of-car" is more likely than the change of perception of the attribute "colour of car" to an entity. In general, more complex concepts are based on finer distinctions and are therefore more volatile.

To accommodate such changes, operators are required to effect any actions consequent to a remodelling. These *evolutionary operators*,

important in any data base system, are imperative in conceptually more complex systems where a detailed perception of the enterprise becomes required. In particular, in an evolutionary system, the usefulness of the additional concepts in the long-term modelling of an enterprise would be greatly reduced without the support of appropriate evolutionary operators.

An evolutionary system would be used in any application in which a large number of interrelated object-types are perceived and where a single object may participate in many different roles.

3.5.6 Further Systems

Two further generalized abstractions, the interval of mapping and the IS-ONE-OF association are not included in evolutionary systems. Their inclusion would make the complexity gap from intermediate systems too large. An evolutionary system represents a buffering step from intermediate systems to advanced systems. The use of the advanced concepts entails a potential decrease in performance as well as an increased effort in data analysis and maintenance. An evolutionary system caters for those applications in which these extra costs are not warranted.

In the move toward more powerful data base systems, generalized constructs need to be developed for those concepts (Sec. 3.3.10) at present perceived mainly in terms of embedded user procedures. Steps in this direction consist of isolating features of these concepts which could be generalized. An evolutionary system containing such features as well as the interval of mapping and IS-ONE-OF association represents an *advanced system* in which all generalized data base abstractions are available.

Of the concepts outlined in Sec. 3.3.10, the representation of time is most significant. Because of the distinctive treatment required,

it is singled out by labelling a system which represents time in a generalized form as a *temporal system*. A facility representing time is conceptually complex and is potentially very costly. A temporal system is therefore placed at the complex end of the spectrum (Fig. 3.15) although it may not contain all possible abstractions.

Although many aspects of the concepts of Sec. 3.3.10 are inherently specific to particular objects, generalizations can be made with respect to applications or specific functions. A simple example is where a derived object representing the count of instances in a specified object set is defined not as a procedure, but as an invocation of a generalized function, COUNT. The choice of which of these functions are to be incorporated into a given system is determined by the intended application of that system.

The point in the spectrum of Fig. 3.15 labelled "further systems", represents those systems which have gone beyond broad generalizations with the further inclusion of characterizations of specialized functions.

3.5.7 Existing Data Base Approaches

Fig. 3.16 illustrates correspondence of abstractions presented in Sec. 3.3 with those of other approaches. The figure indicates authors that consider particular concepts or those similar to it. Where corresponding terms are available, it is indicated in the figure. By necessity, some of the correspondences are approximations. References to the concepts of entity, attribute, relationship and event are not included, as they are compared in Kerschberg *et al.* [1976] and Biller and Neuhold [1977]. It is to be noted, however, that generally these references have not distinguished between relationships and events.

The four concepts of Sec. 3.3.10 are separated into two columns, with time considered individually. The entries for the other three

concepts in the second to last column represent not their generalized treatment, but those authors that have isolated classes of consistency constraints.

Besides the approximate correspondence of the concepts, the sets of concepts proposed in existing approaches consist of unstructured combinations. Furthermore, by the *support* of these concepts, it is meant that any existence and identification considerations inherent in the concepts are system functions. This means that, strictly speaking, data models such as the relational model and the network model do not support any of the concepts. Certainly the mechanisms are available on these models, but the meaning of operations is largely a user responsibility.

Owing to the above considerations, existing approaches cannot be placed exactly into the classes of systems distinguished here. Instead, Fig. 3.17 illustrates the approximate placing of selected approaches in the spectrum.

Binary relational models, such as DIAM and the binary logical association approach, are essentially approaches in fact representation (Sec. 3.2). Only the concepts of entity and relationship can be seen to be modelled, so that they are Basic Models.

Although the semantic network approach is shown at the advanced level, the concepts here are those used mainly for inference (Mylopoulos *et al.* [1976]) in the complementary but distinct field of Artificial Intelligence, and not in data base management. Further, semantic networks and a recent extension, D-Graphs (Weber [1977]), are frameworks in which to construct abstractions as opposed to the isolation of the abstractions themselves. Finally, the temporal systems of Bubenko and the Infological Model do not contain all possible abstractions.

(a)

SUPPLY (SUPPLIER, PART)

SUPPLY (EMPLOYMENT-AGENCY, JOB)

SUPPLIES-TO (SUPPLIER, PROJECT)

(b)

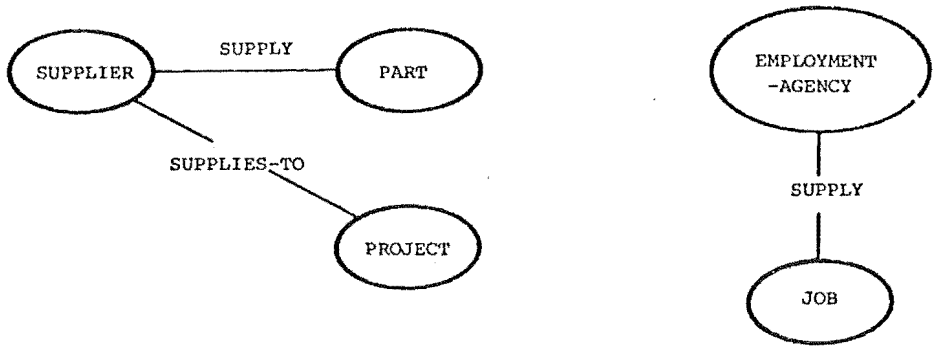


FIGURE 3.1 Fact representation

(a) Predicate representation

(b) Object-oriented representation

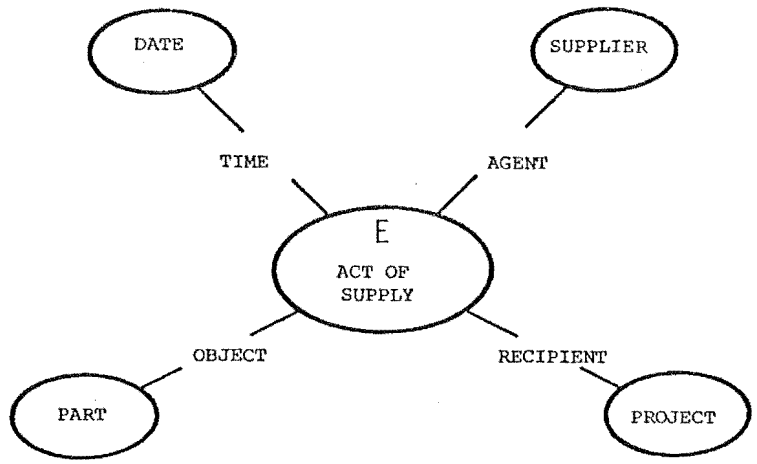


FIGURE 3.2 Object-oriented representation of fact

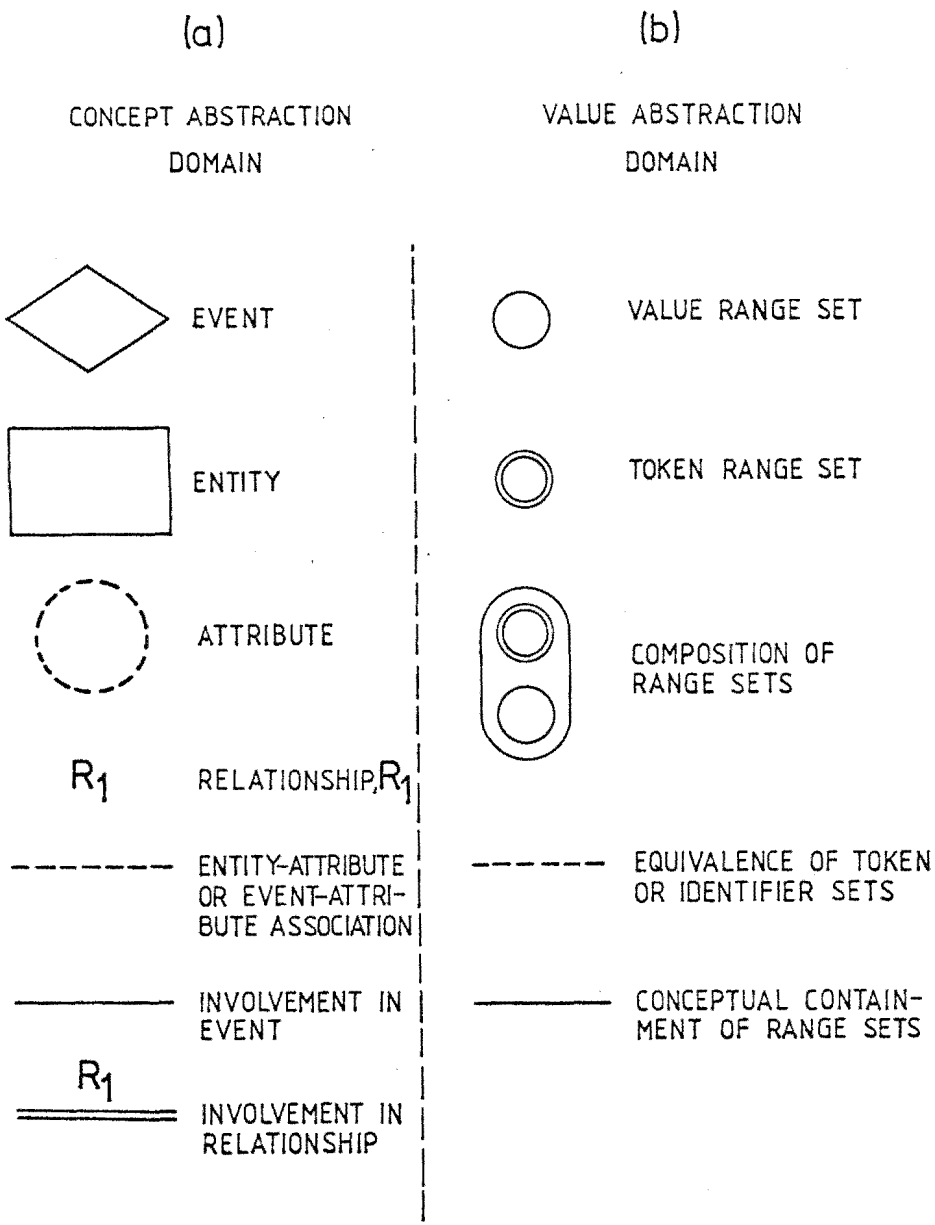


FIGURE 3.3 Abstraction diagrammatic representations
(a) concept abstraction types and their associations
(b) representation of value abstractions

Information to be recorded in a firm:

(1) Projects, their schedule and any special equipment that they may require.

(2) Suppliers and the cities in which the suppliers have major and auxiliary locations.

(3) That projects are supplied by suppliers with particular parts in some quantity.

(4) The regularity of supply of suppliers to projects.

(5) Employees and the projects to which they are assigned.

(6) The dependents supported by the employees.

(7) The names and date of birth of employees and their dependents.

(8) Vehicles owned by the firm.

FIGURE 3.4 Enterprise situation

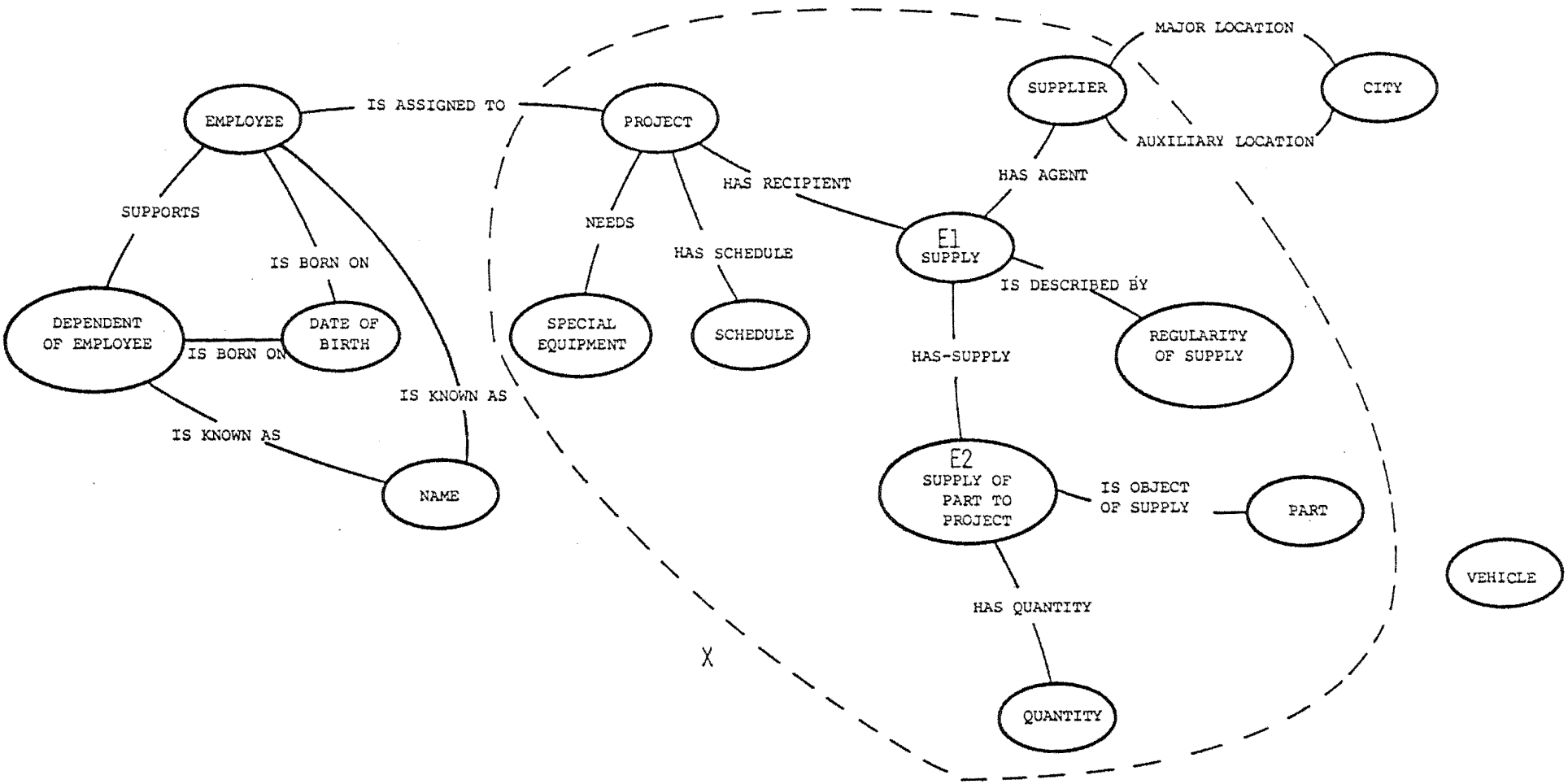


FIGURE 3.5 Binary relational model of enterprise

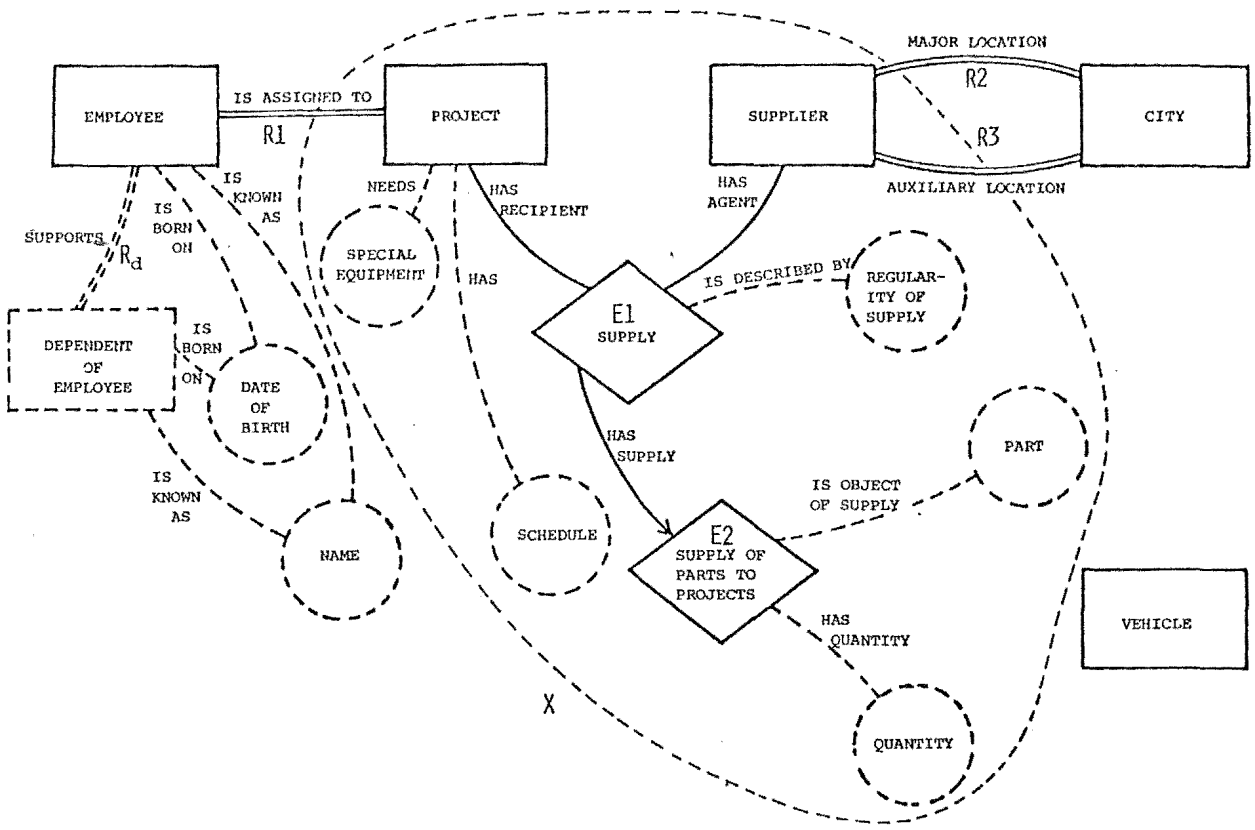


FIGURE 3.6 Abstract model of enterprise

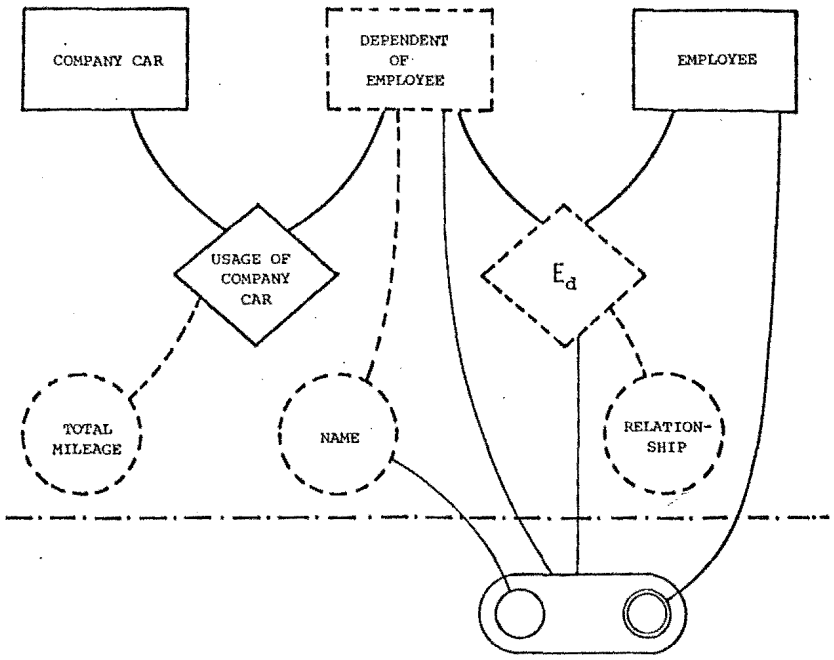


FIGURE 3.7 Dependency objects

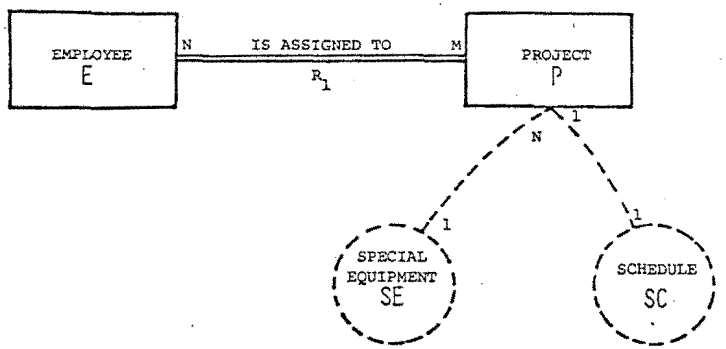


FIGURE 3.8 Cardinality of mapping

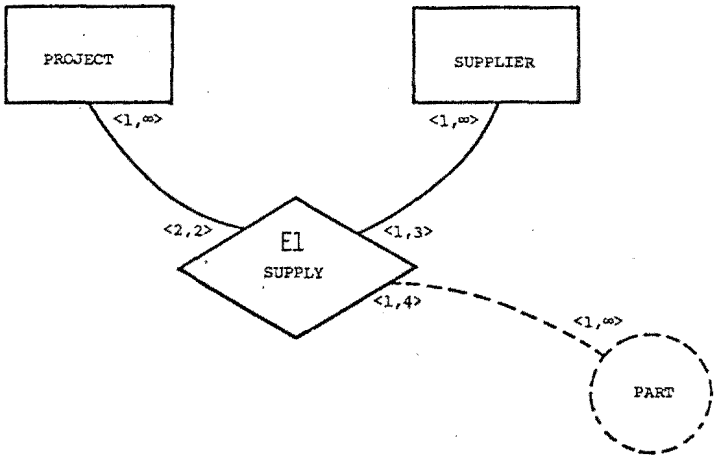
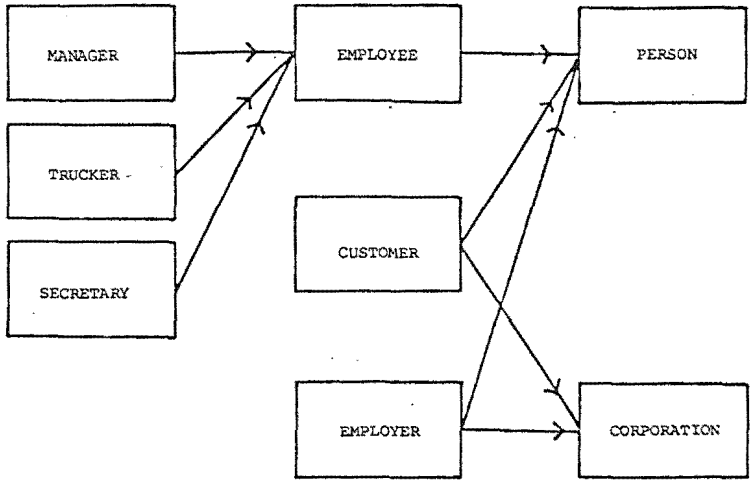


FIGURE 3.9 Interval of mapping

(a)



(b)

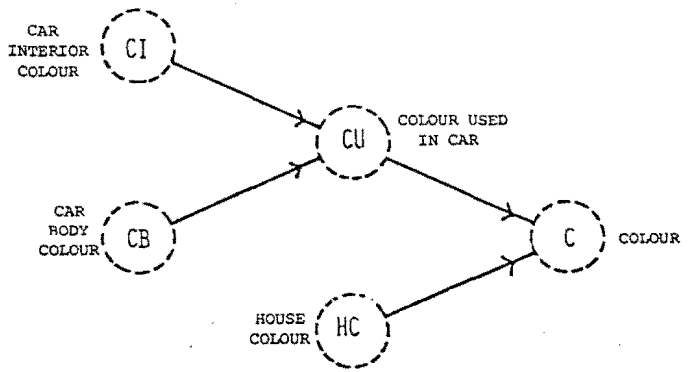


FIGURE 3.10 IS-A Associations
(a) IS-A among entities
(b) IS-A among attributes

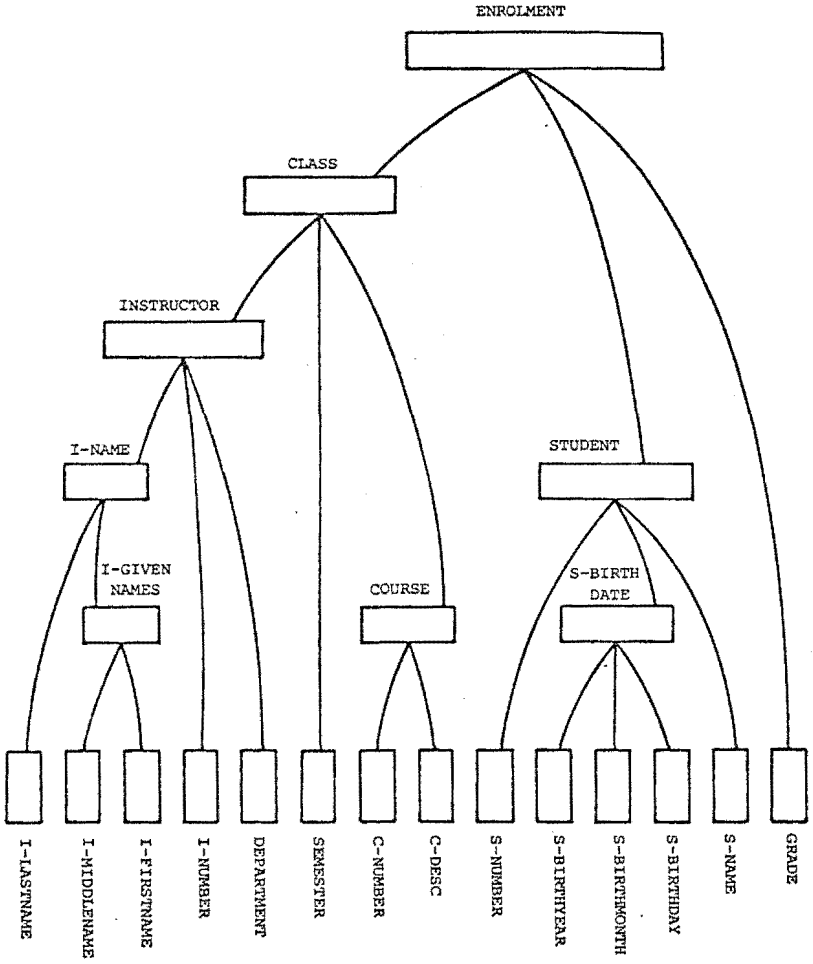


FIGURE 3.11 PART-OF (aggregation) model

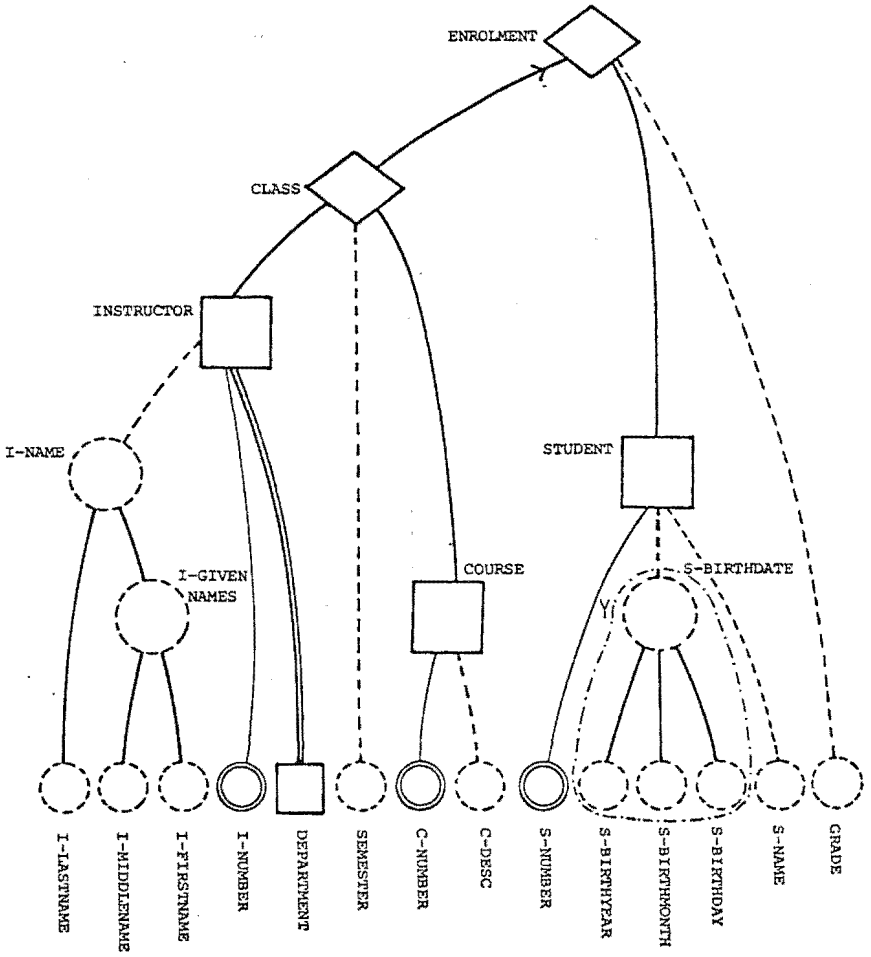
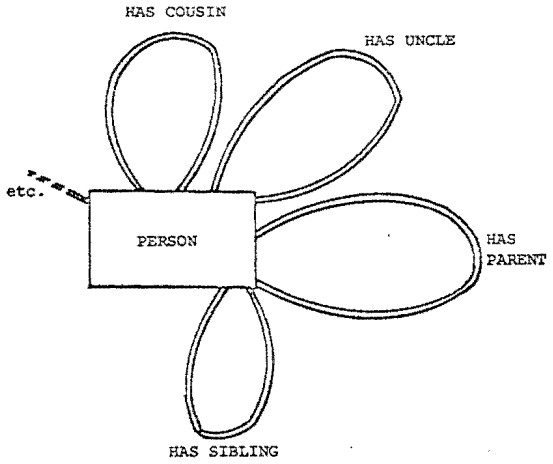


FIGURE 3.12 Abstract model of Figure 3.11

(a)



(b)

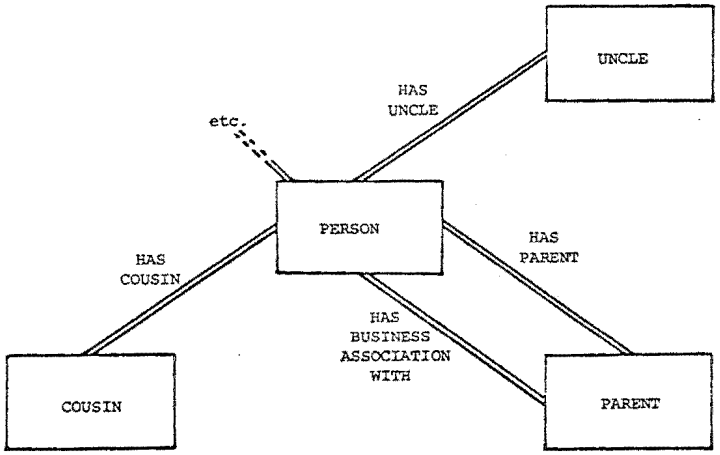


FIGURE 3.13 Association of family-ties

(a) as single-set associations
(b) as sub-object associations

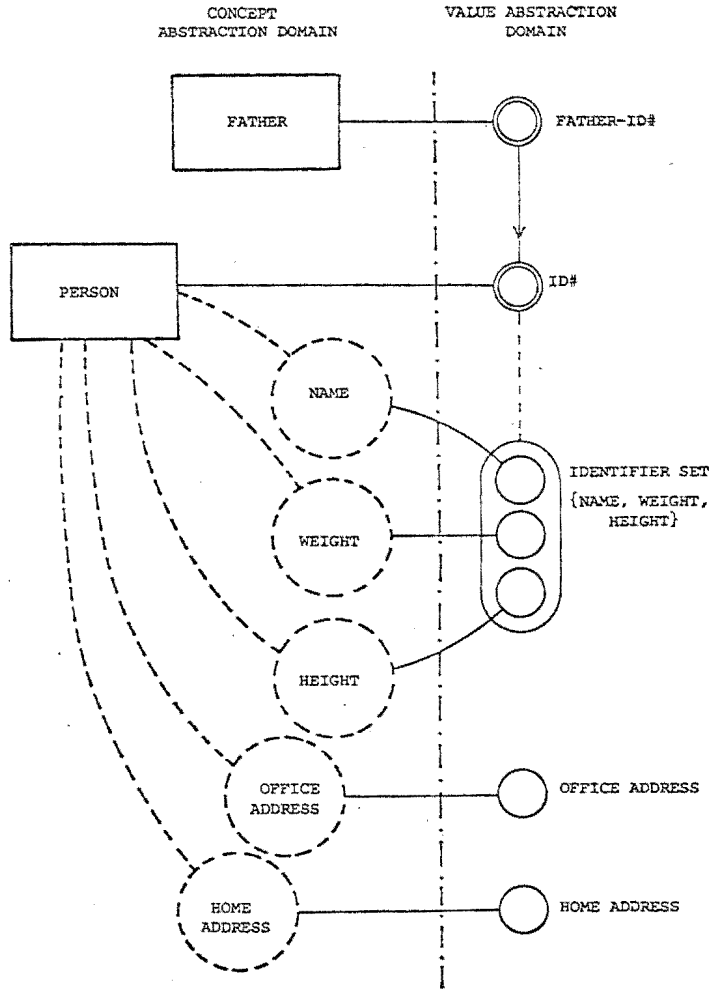


FIGURE 3.14 Value abstraction

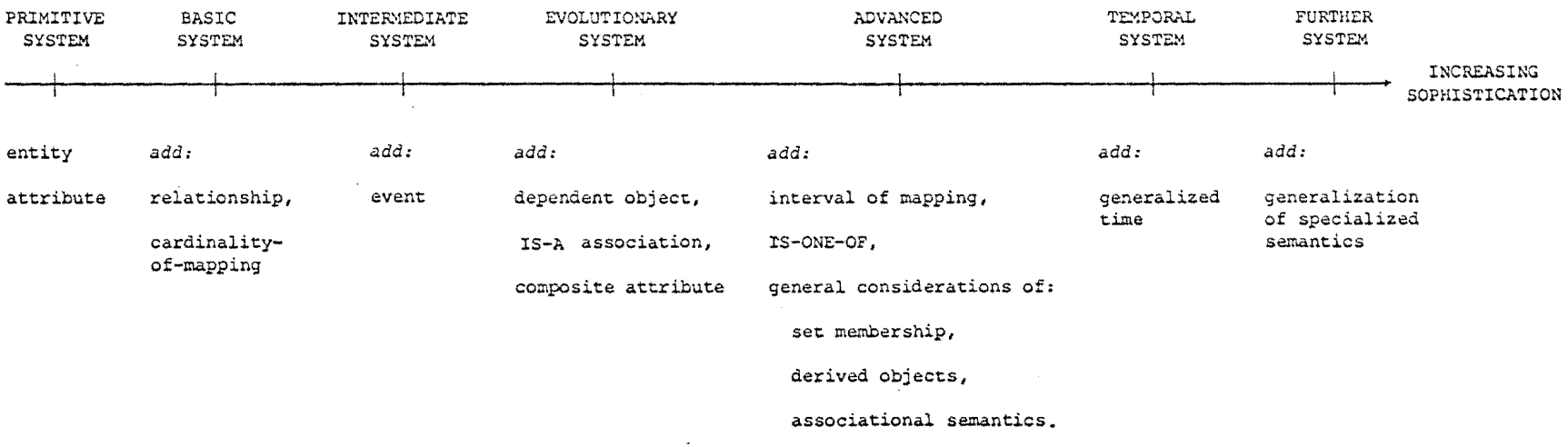


FIGURE 3.15 Conceptual spectrum

AUTHOR	DEPENDENT OBJECT	IS-A/ IS-ONE-OF	COMPOSITE ATTRIBUTE	INTERVAL OF MAPPING	CLASSIFICATION OF PROCEDURAL SEMANTICS	TIME
Abrial [1974]				access function cardinality		considers time
ANSI/X3 /SPARC [1975]		overlapping entity sets				
Benci *et al.* [1976]				constraint of cardinality	integrity constraints	spatio-temporal specifications
Bracchi *et al.* [1976]			internal sets of concepts			
Bubenko [1977]						temporal dimension
Cadiou [1976]		overlapping entities	higher type entities		integrity assertions	
Chen [1976, 1977]	weak entity		split value sets			
Falkenberg [1976]		overlapping types				
Grotenhuis and van den Broek [1976]		role decomposition of info spaces			constraints	versions
Hall *et al.* [1976]		overlapping domains				
Hammer and McLeod [1978]			hierarchically organised attribute			
Hawryszkiewycz [1978]	dependent entity(*Chen)	aggregate entities				
Kerschberg *et al.* [1976]	dependent entity(*Chen)	group/sub/ overlapping entity types		* Abrial		
Machgeels [1976]					integrity constraints	
Mylopoulos *et al.* [1976]		subconcepts	PART-OF			time-related connectives
Schmid and Swenson [1975]	depending part	subconcepts	complex characteristics			
Senko [1977b]		subset-union				considers time
Smith and Smith [1977a, 1977b]		general-isation	aggregation			
Sundgren [1975]						time part in e-messages
Weber [1976]		subset dependencies			dependencies	
Wong and Mylopoulos [1977]	dependency property of objects(*Chen)	* IS-A	* PART-OF	* Abrial	consistency constraints	

"*" is to read "refers to".

FIGURE 3.16 Conceptual correspondence and terminology

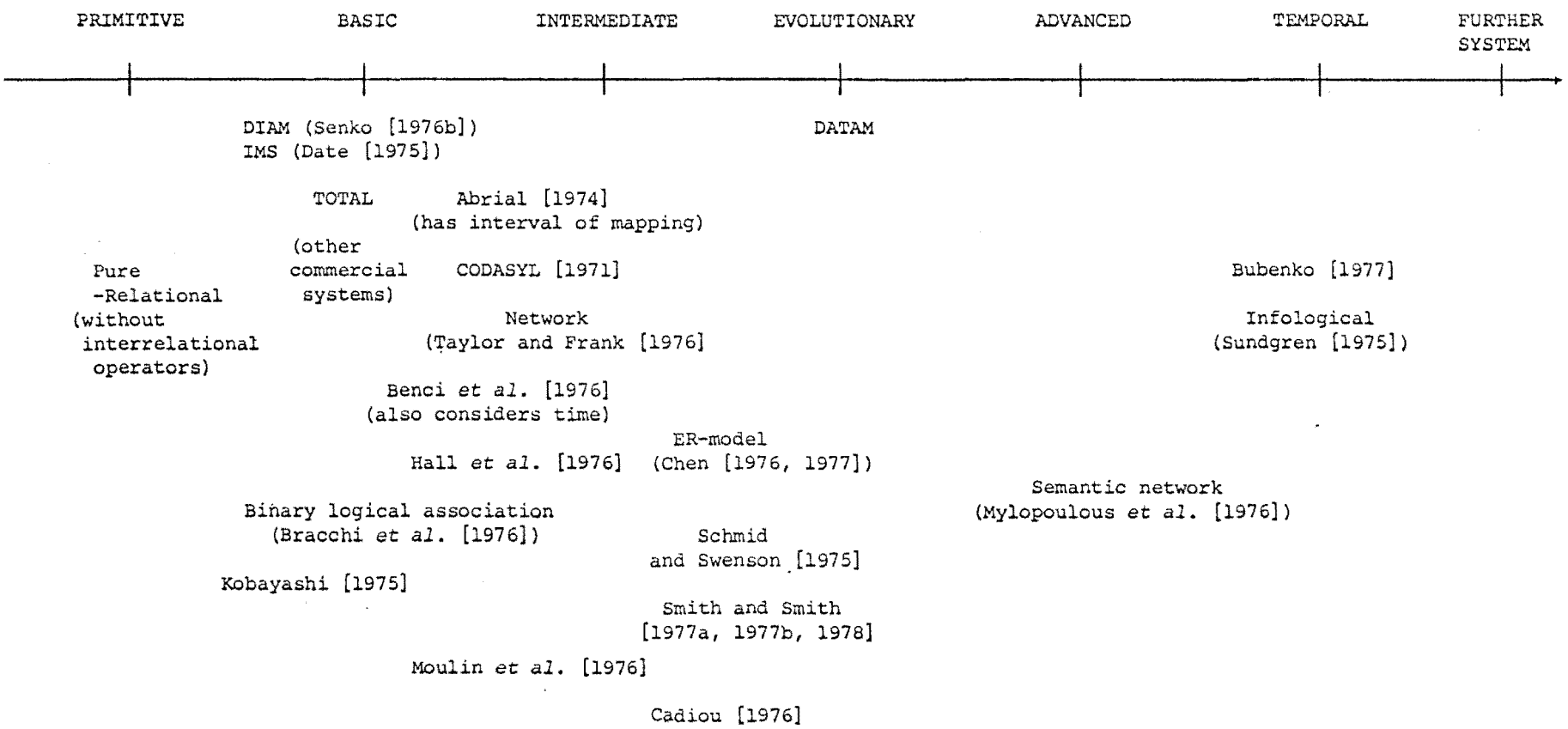


FIGURE 3.17 Approximate conceptual level of existing approaches

CHAPTER 4

DATAM - A DATA ABSTRACTION MODEL

4.1 INTRODUCTION

From the spectrum of Fig. 3.15, it is seen that most existing approaches are either basic or intermediate models (Fig. 3.17). A need is perceived for models at the evolutionary model range of the spectrum. Evolutionary models contain constructs distinguishing real world concepts finer than those of intermediate models. With the aid of suitable evolutionary operators, a data base based on an evolutionary model can evolve easily to reflect finer changes in the perception of the enterprise. As such they would be useful in a large number of applications.

This chapter presents a particular evolutionary model - DATAM (a DATa Abstraction Model). The specifications can be used directly as a basis for an operational system.

Key criteria for community models (Chapter 2) include

simplicity in formulation and understanding
ease of subview formation,
ease of transformation, and
ease of change.

These features are considered in the description of DATAM to support its suitability as a community model.

DATAM is specified in terms of the general framework and abstractions developed in Chapter 3. However, to obtain further simplification, adaptations are made to enable a more natural perception of the enterprise. These are described in Sec. 4.2, in which the

specifications of DATAM are presented.

Modelling examples using the constructs of DATAM are given in Sec. 4.3, including comparisons to other approaches. Examples of the transformation of DATAM views to those in other models are presented in Sec. 4.4. Sec. 4.5 describes subviewing in DATAM.

Concepts supported by an evolutionary system provide the abstractions with which changeable perceptions of reality may be described. Explicit operations are required to perform these changes and any actions consequent to the changes. A complete set of operations for the evolution of an enterprise model based on DATAM is described in Sec. 4.6. Although the operations are given in terms of DATAM concepts, they apply, mutatis mutandis, to corresponding concepts in other models.

Evolution and subviewing are not independent. Ways in which they are related are described in Sec. 4.7.

It is possible to relate abstract models to data models, such as the relational and network models, which are widely documented and whose features are generally agreed upon. Such an association would provide a perspective for specific abstract models relative to existing data base management systems. The use of DATAM in the analysis of the network and relational models is described in Sec. 4.8. This section also describes the construction of a correspondence between the two data models based on DATAM as a semantic reference.

4.2 SPECIFICATIONS OF DATAM

Datam supports the following concept abstractions:

1. Entity
2. Attribute
3. Relationship

4. Event
5. Cardinality of mapping
6. Sub-object association
7. Composite attribute

The other concept abstractions described in Sec. 3.3 are seen to be able to be incorporated in terms of procedural semantics. The value abstractions of Sec. 3.4 are all subsumed by DATAM. Except for the adaptations described in this section, all the specifications and diagrams presented in Sec. 3.3 are adopted in DATAM.

A diagrammatic convention for DATAM IS-A associations is introduced in Sec. 4.2.1. Secs 4.2.2 and 4.2.3 formalize adaptations to the general attribute and event concepts.

4.2.1 DATAM IS-A Representation

Concept abstraction diagrams, such as that in Fig. 3.6, do not include associations of containment among attributes and entities. To preserve clarity, these can be represented separately. In DATAM it is represented as associations of containment among value ranges. The direction of containment is indicated by the relative positions of the ranges, where those further from the concept abstraction domain contain those nearer. Where it is not clear, arrows are used to indicate the direction. However, it is to be noted that the associations among token value ranges pertain to containment of instances, rather than of the range sets themselves.

In Fig. 4.1, the range set for the DATAM attribute (Sec. 4.2.2) Address-of-Phone is shown. For the enterprise (Telephone Company) this is perceived to encompass the Person's (subscriber) Home and Office addresses. The range set "Address-of-Phone" conceptually contains the other two ranges, and this is indicated by the solid lines joining the ranges. Also, the figure shows Staff being a sub-entity of Person as a containment between token ranges.

A value range which conceptually contains others is called a *super-range* set, while ranges which are contained are referred to as *sub-range* sets.

4.2.2 DATAM Attributes

An attribute is defined to be an object which exists only as a descriptor of other objects. The general definition allows a single attribute to be the descriptor of more than one object. This is shown in Fig. 4.2, where the attribute Colour describes both Person and Car. Also, there are two attribute associations between Car and Colour.

This perception, however, does not emphasize the dependence of attributes on entities or the independence of entities from other entities. An attribute cannot be deleted until all associations with it are severed, while the separate viewing of an entity and its attributes would require some agreement concerning any shared attributes. An alternative approach, which emphasizes entities as the centre of interest, is to use *factored attributes*. It is observed that any shared attributes represent generalized concepts which can always be factored into specific concepts involving a single attribute association. An example is given in Fig. 4.3 where the three attribute associations on Colour in Fig. 4.2 are now represented on three separate attributes. The conceptual association between the attributes is evident in that they are all defined on the same range, indicating that they may have overlapping instances. Each attribute is now unshared, which if specified to be permanently so, will always represent a single description of a single entity.

The attribute associations now become less important, since they have a one-to-one correspondence with attributes and consequently do not have to be distinguished. This perception means that the incorporation, manipulation or viewing of description types of an entity need no longer consider other descriptors or entities. It can be determined solely by the entity of interest. This concept of *factored* or *unshareable attributes* appears more natural in an entity-oriented perception. These are the only form of attributes supported by DATAM.

To differentiate this concept from that of generalized attributes the diagrammatic representation of an unshareable attribute is a triangle rather than a dotted circle. Fig. 4.4 shows this for the attributes of Fig. 4.3. Also shown in Fig. 4.4 is the modelling of the attributes on specifically defined value ranges.

A problem encountered in pure binary relational models is the large number of names required (Kerschberg *et al.* [1976]). The approach here allows some economy, since attribute associations now no longer need to be named. Factored attributes are implicit in record-oriented approaches. This factorization of attributes parallels the construction of sub-objects consisting of those instances which participate in particular relationships (Sec. 3.3.10).

The diagrammatic representation of composite attributes as trees (e.g. Fig. 3.12) does not reflect well the physical nature of the composition. In DATAM, a more indicative representation is adopted, in which the figure for a composite attribute encloses those attributes of which it is composed. An example is given in Fig. 4.1, where the composite attribute Date-of-Birth is seen to consist of the attribute Day, Month and Year in that order. Although the representation is not suitable for many-levelled composition, in practice nested composition rarely exceeds two levels.

4.2.3 DATAM Events

A data base event is defined to be the relationship between two objects seen as a describable object. Each db-event reflects the decomposition of real world events into a nested binary relational form. Although such a decomposition is always possible, the necessity of viewing all the basic facts may be a hindrance in the modelling of the real world.

In the example of Fig. 3.6, the event of Supply is justifiably

separated from the event of the Object-Supplied, since the former is described by the Regularity-of-Supply. However, if this event is not so strongly perceived, then such a decomposition is an unwarranted burden. In this case, a more natural perception would be to model the situation as in Fig. 4.5, where E now represents the event of "the supply of parts to projects by suppliers" which is described by its Quantity. This concept is called the *n-way event*. That is, an n-way event may be composed from n ($n \geq 2$) objects. A further example, Fig. 4.6(a), shows a 4-way event representing an association between the entities Project, Supplier, Part and Quantity. This event implies the existence of the corresponding pairwise relationships among the four entities.

The event of Fig. 4.6(a) may be differentiated representationally into either Fig. 4.6(b) or 4.6(c), among other possible diagrams. In Fig. 4.6(b), the event E_2 represents the supply of Part P by Supplier S. This event, together with the entities Project, J, and Quantity, Q, constitute event E_3 representing the supply of P by S to J for amount Q. (The membership of E_2 in the event E_3 is indicated by the arrow.) Fig. 4.6(c) represents an event, E_4, of J-S-P which is involved together with Q in the event E_5.

Each of the diagrams, Figs 4.6(a), (b) and (c) is considered to contain the relationships between the entities, J, S, P and Q. As they are shown, these three diagrams are equivalent conceptually over the totality of instances. Differences, however, become apparent when descriptions for a particular relationship are included. It may be required, for example, to describe the relationship "the supply of P by S" by "regularity of supply". In this case, an association between the event E_2 of Fig. 4.6(b) and an attribute "Regularity of Supply" is formed (as in Fig. 3.6). No other diagrams allow directly for this description. Thus, in the modelling process, Fig. 4.6(a) may be used

when there are no descriptions for the embedded relationships. When such descriptions exist the appropriate model (e.g. Figs 4.6(b), 4.6(c)) has to be chosen. In the case where such a description is perceived and included later, Fig. 4.6(a) may be evolved to the appropriate model to reflect the change in view.

Fig. 4.6 also illustrates the reasons for further entity and event definitions in DATAM. As discussed above, equivalent diagrams involving events and relationships can be constructed for a given number, n, of entities. However, the general definition of an event which allows attributes to be components, would cause inconsistencies in an n-way event. As an elucidation, consider the event of Fig. 4.6(a) but with Supplier and Part being attributes instead of being entities. Then a later possible change to Fig. 4.6(b) results in an event E_2 representing an untenable attribute-attribute (S:P) association. To overcome this representation inconsistency, only describable objects (entities or events) are allowed to be members of DATAM events.

Each member participates in an event in a particular role. If more than one member is of the same object type, then the role of each object has to be specified. On the other hand, the role of objects of different types should be clear from the context of the event. This suggests a saving where, whenever the meaning is clear, object roles need not be named. That is, the links of involvement in DATAM diagrams need not be named. This convention, further alleviates the problem of naming found in binary relational models (Kerschberg *et al.* [1976]).

4.3 ILLUSTRATIVE EXAMPLES

Chapter 3 gives examples illustrating the application of the various abstractions in modelling an enterprise. These examples can

also be seen to describe the use of DATAM concepts, which form a subset of the general approach. However, it is of interest to illustrate further the expressiveness gained by the differentiation of relationships and events, which has not been fully considered in the literature.

This section describes two modelling examples involving relationships and events. For the first example, an analysis of the relational, entity-relationship and DATAM model approaches is included.

4.3.1 Example 1

The following example, from Date [1975] is considered:

The enterprise view concerns Teachers (T), Subjects (J) and Students (S) and their associations. The perceived associations are:

1. For each Subject, each Student of that Subject is taught by only one Teacher.
2. Each Teacher teaches only one Subject.
3. Each Subject is taught by several Teachers.

In terms of functional dependencies, the above statements may be expressed as:

$$S, J \rightarrow T$$
$$T \rightarrow J$$

In the relational model, the representation of the above situation by the relation R(S,J,T) leads to processing anomalies (Date [1975]). Date gives a normalization solution which involves decomposing the relation R(S,J,T) into the relations

R1 (S,T) and

R2 (T,J).

In this case, however, the relations R1 and R2 now require some form of interrelation operators to maintain the semantics which is not indicated by the schema itself. In particular, conditions 2 and 3 above are represented by relation R2, but condition 1 is lost in the schema

and becomes a user responsibility. For example, let the instances of R1 and R2 be as shown in Fig. 4.7. The instances satisfy all conditions. However, in the addition of tuple (S1,T2) to the relation R1, condition 1 (S,J→T) is violated. This violation is discernible only through validation with respect to relation R2 and is not obvious from the relational schema.

Using the entity-relationship diagram (Chen [1976]), a model for the given example is shown in Fig. 4.8. The given conditions 2 and 3 are represented by the ER-relationship R3 and condition 1 is represented by the ER-relationship R4. (The prefix ER- is used to distinguish concepts in the entity-relationship model from the concepts presented in this thesis. Note also that the relationships R3 and R4 in Fig. 4.8 do not represent DATAM events.) Although the entity-relationship model represents the situation better, there is still a semantic aspect in the example that it does not encompass. Let the instances for the ER-relationships be as shown in Fig. 4.8. These instances satisfy the mapping conditions of the entity-relationship diagram. However, the first and fourth instances of R4 reveal an implied violation. An implied condition or consequence of condition 1 is that the Teacher for the Student of a Subject is the Teacher of that Subject. The instances, (J1,S1,T1) and (J2,S3,T1), imply that T1 teaches both Subjects J1 and J2, violating condition 2. The entity-relationship diagram thus does not fully model the given situation.

In DATAM, the semantic constraint in condition 1, that the Teacher involved is the Teacher of that Subject, can be specified explicitly as part of the model. This is possible because DATAM distinguishes between events and relationships, and allows relationships to be specified between two events. The DATAM model for the example is shown in Fig. 4.9. The event E_1 represents the event that students take subjects, and event E_2 represents the event that teachers teach subjects

in a manner (*n:1* mapping) satisfying conditions 2 and 3. The data base relationship R_5 represents the relationship between events E_1 and E_2; that is, for each subject, each student taking that subject (event E_1) is taught by only one teacher of that subject (event E_2). R_5 explicitly satisfies condition 1. Instances for the model are also given in Fig. 4.9.

4.3.2 Example 2

This example involves a supply (Project-Supplier-Part) model. The perceived associations are:

1. A Supplier (S) supplies a Project (J) with a Part (P) in at most one Quantity (Q).
2. Each Supplier will only supply Parts in particular Quantities; therefore, the Supplier in (1) is chosen from a set of Suppliers which supply that Quantity for that Part.

In terms of value associations, these can be expressed as:

1. S,J,P $^{n:1}$ Q
2. Q,P $^{m:n}$ S

The second condition states that the set of allowable Suppliers, S, is determined by the Quantity of the Part required; or, equivalently that the Quantities of Parts that can be supplied is determined by the Supplier (with a *m:n* association of instances).

As with the first example, it is difficult to express this situation accurately with most models. (In particular it leads to a non-BCNF schema (Bernstein [1975]) in the relational model.)

The DATAM diagram for this situation is given in Fig. 4.10. Event E_1 represents the event of Suppliers, S, supplying Project, J with Parts, P. E_2 represents the particular Quantities, Q of Parts. Event E_3 represents the supply of particular Quantities of Parts by a Supplier (condition 2). The data base relationship R_1 represents the

relationship, $E_1 : E_3$ (strictly $E_1 \overset{n:1}{:} E_3$), that the supply of Parts to a Project by a Supplier (Event E_1) is of the correct Quantity (thus satisfying condition 1).

Situations involving multiple associational criteria, such as in the examples, often occur in perceptions of the real world. Such situations are modelled straightforwardly with relationships and events. This expressiveness facilitates accurate modelling of the real world.

4.4 TRANSFORMATION EXAMPLES

This section describes examples of the transformation of DATAM views to views in each of the entity-relationship, binary relational and aggregation/generalisation (Smith and Smith [1977a, 1977b]) models. Procedures are given, based on the diagrams of the models. This is suitable since each model provides diagrammatic conventions. Also, these diagrams provide an effective means of describing the correspondences.

The transformations are illustrated by considering the data base in Fig. 4.11. For clarity, the DATAM diagram does not depict all the value ranges.

The examples show that the transformations are well-defined and are straightforward. Correspondence to the binary relational model is simplest, since this model does not support any abstractions. The most complex is the transformation of a DATAM diagram to an aggregation/generalisation diagram. This is caused by the hierarchical nature of the diagrams. Also, the restrictive constructs of this model necessitate a remodelling of some DATAM objects.

A general feature in all the transformations is that some semantics are lost in the process. This is a result of DATAM being

more powerful in the conceptual spectrum (Fig. 3.17).

4.4.1 Binary Relational Model

The binary relational diagrams considered are those of DIAM II-FORAL (Senko [1976a]). An abstract model is constructed by imposing generalized constructs on a binary model. Therefore to derive the binary relational diagram corresponding to a DATAM diagram involves removing the object interpretations. The procedure given below is defined only on the concept abstraction domain, since the binary relational model does not cater for sub-objects and has no diagrammatic representation for value abstractions. The procedure is:

```
procedure   DATAM-TO-BINARY;
begin
    replace each line (regardless of type) by two solid arcs
            in opposite directions;
    replace all object-type representations by ellipses;
end;
```

The binary relational diagram corresponding to the DATAM diagram of Fig. 4.11 is given in Fig. 4.12. All abstractions are lost. For example, the two sub-objects Non-Academic-Staff and Non-Technical-Staff become objects independent of the rest of the model. Also, the composite attribute Name, all cardinalities and the dependency of Car on Staff, are lost.

4.4.2 Entity-Relationship (ER) Model

In the correspondence to the entity-relationship model any sub-object association is lost. Also, the ER-model represents relationships only as events, so that any DATAM relationship has to be transformed to its event. Further, any relationship or event in which an event is involved has no corresponding structure in the

ER-model.

Attributes in the ER-model are mappings from an entity set to value sets in the "lower conceptual domain" (Chen [1977]). This includes DATAM tokens which correspond to ER-attributes functioning as primary keys. Also, cardinalities of mapping are specified only for ER-entity associations and not for attribute associations. The ER-concept which corresponds most closely to composite attributes is the "split value set" (Chen [1977]).

These factors are taken into consideration in the following procedure:

procedure DATAM-TO-ER;
begin
1. *replace* (i) all lines by single solid lines;
   (ii) all dependent entities by double-lined boxes (ER-weak entity)
2. *delete* any relationship or event in which an event is involved;
3. *retain* (i) all entities and events, which now represent ER-entities and ER-relationships;
   (ii) all cardinalities of entity associations;
4. *replace* all attributes by its value range set, and
   label the attribute association line with the name of the attribute;
   if it is a composite attribute then split the line to each of its component value sets;
   remove all attribute association cardinalities and cardinalities of event roles;
5. *replace* token range sets by single lined circles;
   label the lines to token range sets with the name of the range set;

6. *replace* all DATAM relationships by its event - these then become ER-relationships;
7. *remove* any associations in the value abstraction domain - which now becomes the ER-lower conceptual domain;

end;

The ER-diagram corresponding to the DATAM diagram of Fig. 4.11 is given in Fig. 4.13(a), where the ER-relationships X_1, X_2, X_3, X_4 are events introduced to represent DATAM-relationships. With step (2) in the procedure, the Project entity becomes isolated and the involvement of Academic and Technical Staff is no longer associated to its Projects of involvement. An alternative ER-diagram for this substructure (Staff-Project in Fig. 4.11) is given in Fig. 4.13(b). However, the semantics represented in the two substructures are not the same.

A slight deviation from the procedure is made in the construction of Fig. 4.13(a). The single value set Staff-name is used for the Name attribute of the various Staff subsets. This reflects more closely the typical approach in ER-diagrams.

Further examples of DATAM correspondence to the ER-model are given in Sec. 4.3.

4.4.3 Aggregation/Generalisation Diagram

In the aggregation/generalisation diagram, there are two sets of hierarchies, the generalisation hierarchies and the aggregation hierarchies.

The hierarchies of generalisation correspond to the hierarchies of ranges in the DATAM value abstraction domain. A difference is that the objects themselves are used and not the ranges. This is the basis of the following subprocedure.

```
procedure  CONSTRUCT-GENERALISATION-HIERARCHY;
begin
     comment consider only the value abstraction domain;
     delete all isolated value ranges; all identifier, instance
            composition representations;
     replace each value range by the object defined on it;
          if there is more than one, then each such object
                represents a separate branch in the hierarchy;
     replace each object by a rectangle;
end;
```

The generalisation hierarchies for the DATAM diagram of Fig. 4.11 are given in Fig. 4.14(a). Each lower level node represents *categories* of the higher level nodes. The hierarchy involving Course is not constructed with the above procedure, but in the generation of the aggregation diagram.

The procedure to derive the aggregation hierarchies involves assigning DATAM objects to appropriate levels. Since aggregations represent hierarchies of components, associations requiring roles (e.g. relationship of an object to itself) are not advised (Smith and Smith [1978]). This means that it may be necessary to categorize objects further. Also, additional objects are required to represent *n:m* associations, since aggregation requires a *n:1* correspondence between each object and its components (Smith and Smith [1978]). This means that an aggregation model forces an unnecessarily complex view of the enterprise.

The aggregate objects have components which act as keys. These correspond to DATAM entity tokens, which means that entity token instance sets become primitive aggregation objects in the transformation. Dependencies of objects are lost in the correspondence.

The algorithm to derive the aggregation hierarchies corresponding

to a DATAM diagram is:

procedure CONSTRUCT-AGGREGATION-HIERARCHY;

begin

replace all objects by rectangles; all lines by solid single lines;

replace the nested form of DATAM composite attribute diagrams by corresponding tree forms;

replace any *n:m* association by an aggregate object composed of the two objects associated;

remove all cardinality of mappings;

replace loops by categorizing the object into its role sets and an aggregate object composed of the two created objects;

rearrange the objects into hierarchies using the following algorithm:

1. *put* objects connected by only one line at the bottom of the hierarchy as the primitive objects;

   let these be the objects at level 1;

   i←1;

2. Considering the objects at level i,

   put objects at distance at most 1 from them at level i + 1;

   if there are any objects at level i + 1 connected to objects at the same level then

   for each pair put one object at level i +2, such that the association of the level i + 2 object to that at i +1 is *n:1*;

   i←i+1;

3. *repeat* (2)

   until no objects left;

wherever there is more than one line between two objects:

categorize the lower level object into objects corresponding
to the role of each line;
move the lines to the corresponding new objects;
for each entity which transforms to a non-primitive object
construct a key using:
create a component of the object at the primitive level
whose name is that of the entity's token range set;
if it is a sub-entity, use the range set of the
source entity;
label the objects (including those in the generalisation
hierarchies) to indicate the generalisation level it is at;
comment this is needed because the diagram is not 3-D;
end;

Fig. 4.14(b) shows the aggregation hierarchy corresponding to the DATAM diagram of Fig. 4.11. In this particular example, five objects (those marked 'X') have been constructed to represent *n:m* associations. Not all of these are semantically useful (e.g. Car-Colour) and are merely a consequence of the model. Also, the relationship of Course with itself requires the splitting of Course into two categories and the introduction of a further object (marked 'Y') to represent the relationship.

4.5 SUBVIEWS OF THE ABSTRACT MODEL

4.5.1 Subviews of DATAM Models

Subviewing (Sec. 2.2.2) is defined as the viewing of a subset of the modelled world. This facility is important to the enterprise as groups and individuals within it may want or be allowed to see only a subview of the total view.

Subviewing in data models involves structural formalisations which may result in complexity in the construction of subschema/community schema correspondence (e.g. Biller and Neuhold [1974], Dale and Dale [1976]). In any case, the semantics of the subviewing are based on specific interpretations of the structural representations.

In the data abstraction model, each construct is based on a well-defined associational concept of the real world. Hence, the forming of subviews is determined directly by the meaningfulness of the action on the corresponding concepts of the real world. Thus well-defined rules can be formulated for the construction of subviews. They are therefore easy to understand, which in turn means that subviews become easily specified. The rules for subviewing in DATAM will be described in this section. These rules ensure the formation of a subview that represents a valid and consistent perception of a subset of the enterprise view.

4.5.2 Subviews with Entities

The most basic abstraction is the data base entity which represents the existence of an object in the real world. Since it is meaningful merely to perceive the existence of an entity, i.e. without its descriptors, it is therefore meaningful for a data base entity to exist independently of all other abstractions. Thus, it is consistent to form a subview of Fig. 4.1 consisting of any number of the db-entities, e.g. Person and Phone. There is no logical necessity that these abstraction types be associated in the subview.

4.5.3 Subviews with Attributes

A data base attribute represents a description of a real world object. This means that its existence is meaningful only in association with the data base entity or event which the attribute describes. Also, it is possible to perceive only some of the descriptions of an entity.

Thus, for the DATAM diagram of Fig. 4.1, a valid subview with the attribute Office-Address is one containing this attribute and the entity, Person.

4.5.4 Subviews with Relationships

Data base relationships represent the mutual description between objects, so that in a model the existence of a relationship is meaningful only in terms of the entities or events which it relates. Thus a subview with the relationship R_1 of Fig. 4.1 requires the inclusion of the entities Person and Staff.

4.5.5 Subviews with Events

A data base event requires the explicit perception of the objects involved in the association which the event represents as a describable object. Thus a minimum valid subview containing the event E in Fig. 4.1 requires the concurrent perception of the entities Person and Phone.

Note that the minimum valid subview containing the relationship R_5 in Fig. 4.9 is the complete diagram itself, as the perception of the relationship R_5 requires the perception of the events E_1 and E_2 which in turn require the perception of the entities, J, S and T.

Although only object types are considered, the rules of subviewing given in this section apply as well to instances. However, in general, the support of type subviewing and instance subviewing involves different considerations.

4.6 EVOLUTION OF THE ABSTRACT MODEL

4.6.1 Evolution of DATAM Models

The construction of an abstract model represents the perception of the real world at the time of construction. As perceptions of the real world change with time, changes must also be made to the abstract

model to reflect these changes in perception, in order to maintain the usefulness of the data base (Swartout *et al.* [1977], Navathe and Fry [1976], Chen [1977]).

Except for Chen [1976, 1977], evolutionary operations of abstract models have not been rigorously treated in the literature. Chen describes operations in an intermediate system (Sec. 3.5) environment consisting of the entity, attribute, and relationship concepts of the ER-model. Other references consider evolution in terms of data model restructuring (Swartout *et al.* [1977], Navathe and Fry [1976]). This section introduces further operations that are necessary for the evolution of the abstract model to reflect any perceived changes.

Changes to the abstract model may result in an increase or decrease in the number of concepts modelled; and the evolution process is correspondingly defined as *progression* or *regression*. Also, some changes, such as changing an object's name, are not seen to increase or decrease the number of concepts modelled. These changes are said to perform a *modification* on the model. Fig. 4.15 lists progression/ regression and modification operations for the evolution of DATAM models. These operations are described in the following sections.

4.6.2 Entry and Exit of Objects

Entry and exit operations are required for each of the objects provided by the model, which in DATAM are entity, attribute, relationship, event and range. There are two sets of operations, one each for object types and instances. In general, the rules or consequential actions in both type and instance operations are the same.

Operations for the entry and exit of ER-objects have been discussed by Chen [1976, 1977]. DATAM rules for these operations are similar and are described in the treatment of existence dependencies in

Chapters 3 and 4. Consequently, they are not considered further here.

4.6.3 Factorization and Consolidation of Object Types

Factorization and consolidation operations are those which involve the subsetting or merging of objects existing in the data base. Four forms are discerned. These are listed in Fig. 4.15 and described in the following subsections. As factorization and consolidation operations are mutual converses, it is sufficient to consider one of them. The following discussion will be limited to the factors involved in factorization operations.

4.6.3.1 Independent objects

Given an entity type, it is possible to derive a subset of its instance set dynamically, using appropriate excess operators. Such a subset can then be perceived as the instance set of a further object-type in the model. Upon construction, this new object type need not be associated with the source object in any way. That is, the objects can develop independently. Such sub-object construction may be defined for each of the object types of entity, attribute, relationship and event.

Similar operations of Swartout *et al.* [1977] and Chen [1977] are essentially of this form, since there are no provisions in these approaches to specify subset dependencies between object sets. In DATAM, dependent sub-objects can be constructed and are considered in the following two sections.

4.6.3.2 Role factored sub-objects

Using IS-A associations, it is possible to specify that the instance set of a given entity is necessarily contained in the instance set of another entity. The object thus created is a sub-entity-type.

Such sub-objects can be differentiated on their manner of construction. In particular, the formation of sub-objects can be based

on their roles in specific associations. An example is given in Fig. 3.13, where in Fig. 3.13(a) Person is shown to have various relationships with itself. In Fig. 3.13(b), objects have been constructed for each relationship, where each object contains those Persons which participate in a particular relationship. These *role factored* objects can then be specified as sub-objects of the source entity Person.

Another example is where a subset of an entity set becomes perceived as participating in a relationship with a cardinality different from that existing. This requires the construction of another relationship having the correct cardinality and defined on the sub-object corresponding to the subset. This is illustrated in Fig. 4.16, where the diagram of Fig. 4.16(a) is evolved to that in Fig. 4.16(b). Here, external information is required to specify the subsetting.

Factorization with respect to roles is applicable on entities and events. Note that DATAM attributes are already factored (Sec. 4.2).

4.6.3.3 General sub-objects

In the general case, any existing object type can be made to be a sub-object of another object. However, the objects may have been constructed with respect to arbitrary criteria. Therefore, the specification of an object as a sub-object of another requires tests to ensure that their instances conform to the specifications.

With entities, this involves checking that the sub-object instance set is contained in that of the source entity. On the other hand, for an attribute to be a sub-object of another, then its value range must be contained in that of the other.

4.6.3.4 Composition of instances

The previous three sections concern factorization with respect

to sets. Similar operations are also required on instances to evolve the two DATAM concepts of composite attribute and identifier.

The operations necessary to effect changes to the composite-attribute are operations for the addition of attributes to and deletion of attributes from a composite-attribute. These operations include those for the composition of a composite-attribute from a set of attributes and the corresponding decomposition. Although specified in terms of attribute types, the operations in fact define the restructuring of the instances.

Evolution of identifiers also requires operations to add attributes to and delete attributes from an identifier.

4.6.3.5 n-way events

The concept of the n-way event (Sec. 4.2) allows economy in the perception of the real world. To allow for subsequent refinement in the perception of events, operators are required to evolve n-way events into its constituent events.

An example is given in Fig. 4.6, where Figs 4.6(b) and 4.6(c) are two alternative factorizations of the 4-way event of Fig. 4.6(a). Operators to perform such changes have to account for any association the n-way event may have. For example, if the 4-way event E_1 of Fig. 4.6(a) is described by an attribute, then in the alternative perceptions this attribute becomes the attribute of E_3 and E_5 of Figs 4.6(b) and 4.6(c) respectively. This is because these two events represent the association among the four entities corresponding to that in E_1.

The reverse operation, in which events become consolidated, is seen as a regression of the enterprise model.

4.6.4 Transformation of Abstraction Types

As the perception of an enterprise changes, an object previously

modelled as one abstraction type may become perceived as being of a different abstraction type. Such transformations of abstractions for DATAM are listed in Fig. 4.15. The transformation from left to right represents a progression, and right to left a regression. As each transformation is reversible, the following discussion will be limited to progression operations.

4.6.4.1 Attribute → entity

A db-attribute represents the description of an entity. If the descriptor (attribute) itself becomes perceived as a db-entity, then the previous entity-attribute association becomes a relationship, representing the mutual description between the entities. This change in perception has repercussions on the underlying range sets as the change of an attribute to an entity means that its value range set becomes a token range set. An example is given in Fig. 4.17. When the attribute, Address-of-Phone, becomes an entity (with its own attribute, Category-of-Address), the Phone : Address-of-Phone association becomes a relationship, R_2, and the Address-of-Phone value range set becomes a token range set.

Such a change may also force the change of other attributes into entities. This would occur if the attribute being changed represents a description which conceptually encompasses others in the data base. Hence where the value range of the attribute being changed is a super-range set, then the contained ranges also become token ranges. This in turn changes the attributes defined on them into entities and their associations into relationships. In Fig. 4.17, for example, the change in perception of the attribute Address-of-Phone into an entity forces a change in the other two attributes (Office-address and Home-address), which are conceptually encompassed by the Address-of-Phone attribute, into entities.

A similar change also occurs when a sub-range set becomes a token

set as this implies that its super-range set has also to be a token set. For example, the evolution of the attribute Home-address in Fig. 4.17 into an entity would also result in the change of the attributes, Address-of-Phone and Office-address, into entities.

In other words, the hierarchy of IS-A associations among the attributes becomes transformed to that among entities. However, the IS-A associations among attributes do not extend to containment of instance sets. This means that in the evolution operation, appropriate instances have to be propagated through the instance sets to ensure the containment required by entity IS-A associations.

4.6.4.2 Entity → event

An event (object) of the real world may be modelled as a db-entity. For example, in Fig. 4.17 the real world perception of a "marriage" event is modelled as a db-entity (Marriage). If the objects involved in the real world event are themselves modelled and their associations with the db-entity which represents the event are to be indicated, then this db-entity is changed to a db-event. This progression is illustrated in Fig. 4.17 where the couple involved in the marriage is to be indicated.

In this progression, external information is required to associate the tokens of the resultant event to its involved entities. In the example above, for each Marriage event (represented e.g. by its licence number), the two Persons whose Marriage it represents have to be specified in the evolution.

4.6.4.3 Relationship → event

A data base relationship represents the association or mutual description of two db-entities or events. If this relationship is to be viewed as an object or is to be described, then it is modelled as an event. This change is shown in Fig. 4.17 where relationship R_1 becomes event E. This operation requires the creation of a token range

set for the resultant event. As the associated values from the token instance sets of the objects involved in the event are always identifiers of all instances of the event, a token range set of the event can therefore be defined as the concatenation of the values of the token range sets of the objects involved. For example, Fig. 4.17 shows the token range set for the event E as being made up from the token range sets of the entities Person and Skill. Thus, where the event token is to be defined by the member objects, then no explicit token need be declared.

A similar change occurs when an entity-attribute association is to be described. The attribute is first changed to an entity (Sec. 4.6.4.1) and the resulting relationship then changed to an event.

The progression of a dependency-relationship to a dependency-event is also similar and is discussed in Sec. 3.3.6.

4.6.4.4 Dependent entity → regular entity

A dependent entity is one whose existence and identification are possible only through its owning entity with respect to a corresponding relationship. The change of a dependent entity to a regular one means that the entity is to exist and be identifiable independently of other objects. This, simply involves the creation of a token range set for the dependent entity to reflect the corresponding change of the dependency-relationship to a regular one. For example, in Fig. 3.6, a perceived change of the dependent entity, Dependent-of-Employee, to a regular one would be effected by the creation of a token range set for this entity. The dependency-relationship, R_d, in Fig. 3.6 then becomes a regular relationship.

4.6.5 Modification Operations

Fig. 4.15 lists four classes of modification operations, which are discussed in the following subsections.

4.6.5.1 Change of range attributes

Range attributes such as type and length, are used to specify the form and scope of the values of object instances. Changing an attribute of a range, for example its (character) length, typically involves physical manipulation of existing instances to conform with the new specification. Such manipulation is usually subject to specific conventions. For example, if the length of a range is increased, then a particular convention may be to pad existing instances with trailing blanks.

4.6.5.2 Change of object names

As with other aspects of real world modelling, the naming of object types is subject to change. Operators are required to provide for such changes.

The corresponding operation for instances is the changing of the value of instances. An example is where a Person changes address, so that the value of the instance of the address attribute of that Person has to be replaced by the new value.

4.6.5.3 Change of mapping cardinalities

In DATAM, the cardinality of an association may be *1:1*, *1:n*, *n:1* or *n:m*. A change in the perception of the cardinality of any association has to be reflected in the data base. If the initial cardinality is *1:1* or the change is from *1:n* to *n:m* or from *n:1* to *n:m*, then no changes are required on existing instances. Any other changes, however, require checking of existing instances and some convention in the deletion of instances that may be required to conform with the new specification.

4.6.5.4 Change of ordering in instance composition

The concepts of composite attribute and identifier concern the composition of attribute instances. This composition is typically perceived as the physical concatenation of the instances. The order

in which this concatenation is to be done is subject to change, thus requiring some operator for this purpose.

4.7 EVOLUTION AND SUBVIEWING

Evolution and subviewing are related in various ways. This section discusses the effect of evolution of the community view on a subview, and the implications of using generalised regression as the basis for subview construction.

Since a subview is defined with respect to a specific enterprise model, its consistency is determined by the state of the model at the time of the subview construction. However, subsequent evolutions may affect the consistency of the subview representation as a subset of the model. That is, the subview may contain associations or objects not now represented by the data base.

The level at which a subview is specified has a bearing on the actions required to maintain consistency in the event of an evolution (particularly, a regression) of the enterprise view. Subviews can be specified at two levels, one where the subsetting is of only the object types and the other where the subsetting also includes that of the instances. In the first, the subview would require changes only when regression of object types occurs on the model, while in the second, instance changes may also need to be propagated to the subview.

The choice of the level of definition of subviews is determined by the degree of awareness required by the user. It is usually sufficient to adopt the first approach discussed above, and construct subviews as subsets of the concept abstractions with sharing of the instances.

The construction of subviews described in Sec. 4.5 could be effected by regressions involving the exit of objects. In general,

any form of the regressions implied by Sec. 4.6 results in a conceptual subset of the enterprise view, and can therefore be used to construct subviews. However, constraints need to be made on the use of subviews constructed by such generalized regressions. In particular, the objects in a subview may now be of concepts which are of lower degrees of interest than the corresponding object in the enterprise view (e.g. an entity may be viewed as an attribute). This means that the dependencies of existence in the subview may be in conflict with that of the enterprise view.

The resolution of this conflict involves more than the resolution of identification and access differences which are necessary in any subviewing. However, it could be avoided by disallowing any deletions based on a subview to be propagated to the enterprise view. To allow deletions would require the formulation of specific rules on the intended interaction. This constraint on subviews could be extended to disallow any evolution of the subview to be effective on the community view.

4.8 ANALYSIS OF DATA MODELS

4.8.1 DATAM as a Semantic Reference

Since data models are based on representation structures they are subject to interpretation as it is not clear what concepts are represented. In the analysis of data models, an interpretation can be fixed in terms of an external semantic reference. Abstract models can provide such a reference. In general, an interpretation of a data model imposes a discipline on the construction of its structures. The use of DATAM in the analysis of data models is presented in this section.

As data model structures are representations of instances of associational concepts, correspondence with DATAM is best seen through

comparisons with DATAM instance representations. Sec. 4.8.2 describes the representations that will be used. Analyses of the relational and network models are presented in Secs 4.8.3 and 4.8.4 respectively. These analyses form the basis for the consideration of correspondence between the models, which is described in Sec. 4.8.5.

4.8.2 A Representation of DATAM Instances

A perception of DATAM instances is required in the analyses of data models. In order to relate to the structural representations of the relational and network models, a table form is used to represent DATAM associational instances. Different DATAM associations are represented by different table types.

4.8.2.1 EA-table type

Instances of the associations of an entity to its attributes are represented by an EA-table (Fig. 4.18(a)). In this table, the first column represents a token instance set of an entity type and the other columns represent the instance sets of all the attribute types of that entity. Each row of the table indicates a particular association of the entity instance to its attribute instances. For example, the first row in Fig. 4.18(a) represents an association of a token t_1 of the entity to particular attribute instances, e.g. v_{11} of attribute type A_1. Row two represents an association of the same token t_1 with another instance, v_{12}, of the attribute A_1.

4.8.2.2 R-table type

Tables of this type represent relationships. Each table (e.g. Fig. 4.18(b)) consists of two columns containing token instances of the two participating object-types (entities or events).

4.8.2.3 Ev-table type

These tables represent events. Each table has columns representing the event tokens, tokens of the member objects, and the attribute

instances of the event. These represent the existence of the event, the embedded relationships among the member objects (see Sec. 4.2.3), and the descriptions of the event. An Ev-table is illustrated in Fig. 4.18(c).

4.8.3 Analysis of the Relational Model

In the relational model (Codd [1970]), there is only one structure - the relation. The construction of relations based on functional dependency constraints results in normalized relations (Codd [1971b], Bernstein [1975]). These normalized relations have a correspondence with the tables in the DATAM representation of Sec. 4.8.2. This fact is used in the analysis of normalized relations where relations structurally similar to one of the DATAM tables are interpreted as representing the same concept as that of the table.

No direct correspondence, however, can be made between relational keys and DATAM tokens. The keys in the relational model uniquely identify tuples (rows of the tables) rather than db-entities. It is thus difficult to base a semantic analysis of a relational model on the internal structure of its relations as this reveals little of the represented concepts of the real world.

The stated functional dependencies of a relational model represent the perceived semantic constraints and they completely determine the construction of normalized relations. An analysis of a relational model can therefore be based on functional dependencies. Nevertheless, because functional dependencies are defined on representational rather than abstract structures, such interpretation is subject to ambiguity. This ambiguity can be resolved by relating each distinct dependency type to an abstract concept.

A set of dependencies (Bernstein [1975]) for a relational model is given in Fig. 4.19(a). The set of normalized relations constructed

from these dependencies, as given by Bernstein, is shown in Fig. 4.19(b) and the DATAM interpretation of the model is presented in Fig. 4.20.

4.8.3.1 Descriptor types

As stated, an analysis of a relational model here is based upon its dependencies. Consider the dependency, T : Stock# → Price, of Fig. 4.19(a). The dependency represents the instance associations of Stock# and its Price. Here Price can be viewed as a descriptor of Stock#, that is, the Right Hand Side (RHS) of this dependency is a descriptor of the object on the Left Hand Side (LHS). This corresponds to a db-entity : db-attribute association so that the dependency T can be interpreted as the DATAM substructure T_d in Fig. 4.20. T_d represents the Stock $^{n}:^{1}$ Price association where the entity Stock has Stock# as a token. The relational dependency W is similarly interpreted as the DATAM substructure W_d.

Although the RHS of a dependency (such as T or W) is typically the descriptor of the LHS, this situation is not always so. For example, if the instance association for Stock# : Price is *1:n* (instead of *n:1*), then T' representing this dependency would be expressed as T' : Price → Stock#, where the descriptor (Price) is now on the LHS. To keep the descriptor on the RHS, a notational change for T' to T' : Stock# ← Price could be made. This suggests a worthwhile notational rule for dependencies where the descriptor types are clearly identifiable. In the interpretation of Fig. 4.19(a) it is assumed that the RHS always represents a descriptor.

4.8.3.2 Entity type

In DATAM, a descriptor which is itself described is a db-entity. In the consideration of the dependencies U and W of Fig. 4.19(a), City is a descriptor of Dept (from dependency U) and is itself described by Population (dependency W). Therefore City is to be viewed as a db-entity. Thus the Dept : City association is one between two entities,

and is the db-relationship shown in the DATAM substructure, U_d, in Fig. 4.20.

4.8.3.3 Event type

Situations occur where either side of a dependency consists of more than one object (relational attribute). This combination represents an association among the constituent objects and is interpreted as a db-event. For example, the dependency S has the combination {Stock#,Dept#} which is interpreted as the db-event, Stock-in-Dept, involving the entities, Stock and Dept. The dependency S therefore represents the description of the event by the db-attribute, Quantity, and is shown as substructure S_d in Fig. 4.20.

4.8.3.4 Multivalued dependency

From the above analysis, functional dependencies are seen to represent associations between objects and their descriptors. The definition of functional dependency (Codd [1971b]) allows only for the specification of *n:1* associations. (This is not to say that the relational model cannot represent other associations - only that it is not explicitly specifiable at the level of dependencies.) To provide for the specification of *m:n* associations between objects and their descriptors, Fagin [1977] recently introduced the generalized *multivalued dependency*. In the interpretation of general dependencies, each DATAM association (*m:n* or *n:1*) follows that of the dependency.

4.8.4 Analysis of the Network Model

The network model has two basic structures - record type and set type. Network structures are generally represented by data structure diagrams (Bachman [1969]).

4.8.4.1 Entity-attribute associations

A record type is used to represent the existence and description of a real world entity (Taylor and Frank [1976]), so that, in value instances, it corresponds to the DATAM EA-table of Sec. 4.8.2. The keyfield of this record type corresponds to the token column of the

EA-table and non-key fields correspond to attribute columns. Thus a record type can be interpreted as entity-attribute associations where a repeating group is viewed as an *n:m* entity-attribute instance association, and a field within the record type is viewed as an *n:1* association.

Fig. 4.21(a) shows a network model.

The non-key fields for each of the record types are also indicated in the figure. Record types are interpreted as db-entity : db-attribute associations, so that the record type City corresponds to the DATAM substructure W_d in Fig. 4.20. Similarly, record type Stock corresponds to the DATAM substructure, T_d.

4.8.4.2 Relationship type

A set type represents the *1:n* association of record occurrences of a record type to those of another. In value instances, therefore, a set type corresponds to the DATAM R-table with an imposed *1:n* association restriction.

The set type, Location, in the substructure U_n in Fig. 4.21(a) represents the *1:n* association of City occurrences to Dept occurrences. This is interpreted as the db-relationship between the db-entities City and Dept in Fig. 4.20.

4.8.4.3 Event type

Consider the substructure, S_n. In the correspondence to DATAM, network structures of this form are ambiguous since they can be interpreted as representing two distinct concepts (Chen [1976] also notes the semantic ambiguity of such network structures). In one case, the record type Stock-Dept-Link is seen as a db-entity so that the set types Held-in-Dept and Stock-Held are seen as two distinct db-relationships involving this entity.

On the other hand, the record type, Stock-Dept-Link, can be seen to facilitate a many-to-many association of the record types, Stock and Dept. In this case, the substructure, S_n, is interpreted as the DATAM substructure, S_d, containing a db-event.

To resolve the above ambiguity, in the analysis of a network model, the interpretation of network structures of the form of S_n has to be fixed (see also Chen [1976]). One method is to adopt the second approach discussed above. This involves the interpretation of record types that are owned as forming db-events. With this approach, however, the record type, Dept, in Fig. 4.21(a) would become interpreted as a db-event rather than as a (intended) db-entity. To overcome this, the association between Dept and City has to be represented through the substructure, X_n, shown in Fig. 4.21(b) where a dummy record type is introduced to facilitate the association. With this change, the relationship between the entities, Dept and City, becomes embedded in the db-event, Dept-City-Link. Therefore, in this method, there is no network structure that can be directly interpreted as a db-relationship.

4.8.5 Correspondence of Data Models

The transformation of a data model to another based on their structural constructs (e.g. McGee [1974]) is semantically intuitive (Biller and Neuhold [1978]) and therefore provides no guarantee of the validity of the correspondence where the interpretation of particular structures may be ambiguous. To ensure a valid correspondence, a semantic reference is used to impose an interpretation on the models with respect to which the correspondence is to be defined. DATAM provides a basis for such an interpretation.

This section discusses the correspondence between the network and relational models in terms of the DATAM interpretations presented in Secs 4.8.3 and 4.8.4. In the interpretation of the network model presented in Sec. 4.8.4, a discipline was imposed where dummy record types are introduced in order to preserve the entity role of some record types. A consequence of this is that db-relationships have no distinct network representation, and therefore transformations between relational

and network models may not be reversible.

4.8.5.1 Relational to network transformation

This involves the translation of any given set of relational dependencies into an equivalent network structure. Consider, for example, the dependencies in Fig. 4.19(a). A DATAM interpretation of this relational model is given in Fig. 4.20. The network representation of this DATAM model is that of Fig. 4.21(b). Therefore Fig. 4.21(b) is the corresponding network model for the relational model of Fig. 4.19(a).

4.8.5.2 Network to relational transformation

The reverse transformation of the network model of Fig. 4.21(b) does not result in the relational dependencies of Fig. 4.19(a). Specifically, the substructure, X_n, cannot be interpreted as the DATAM substructure U_d in Fig. 4.20. The DATAM interpretation of X_n contains a db-event rather than a db-relationship. This DATAM interpretation, being different from that of Fig. 4.20, will not produce the set of dependencies of Fig. 4.19(a). Instead, it corresponds to the set of dependencies of Fig. 4.22, in which the notation {Dept#, City} $\rightarrow \phi$ (Fagin [1977]) indicates that there are no relational attributes which are dependent on the key {Dept#, City}.

A reversible transformation is possible if the relational model involved does not contain dependencies which become db-relationships in the DATAM interpretation.

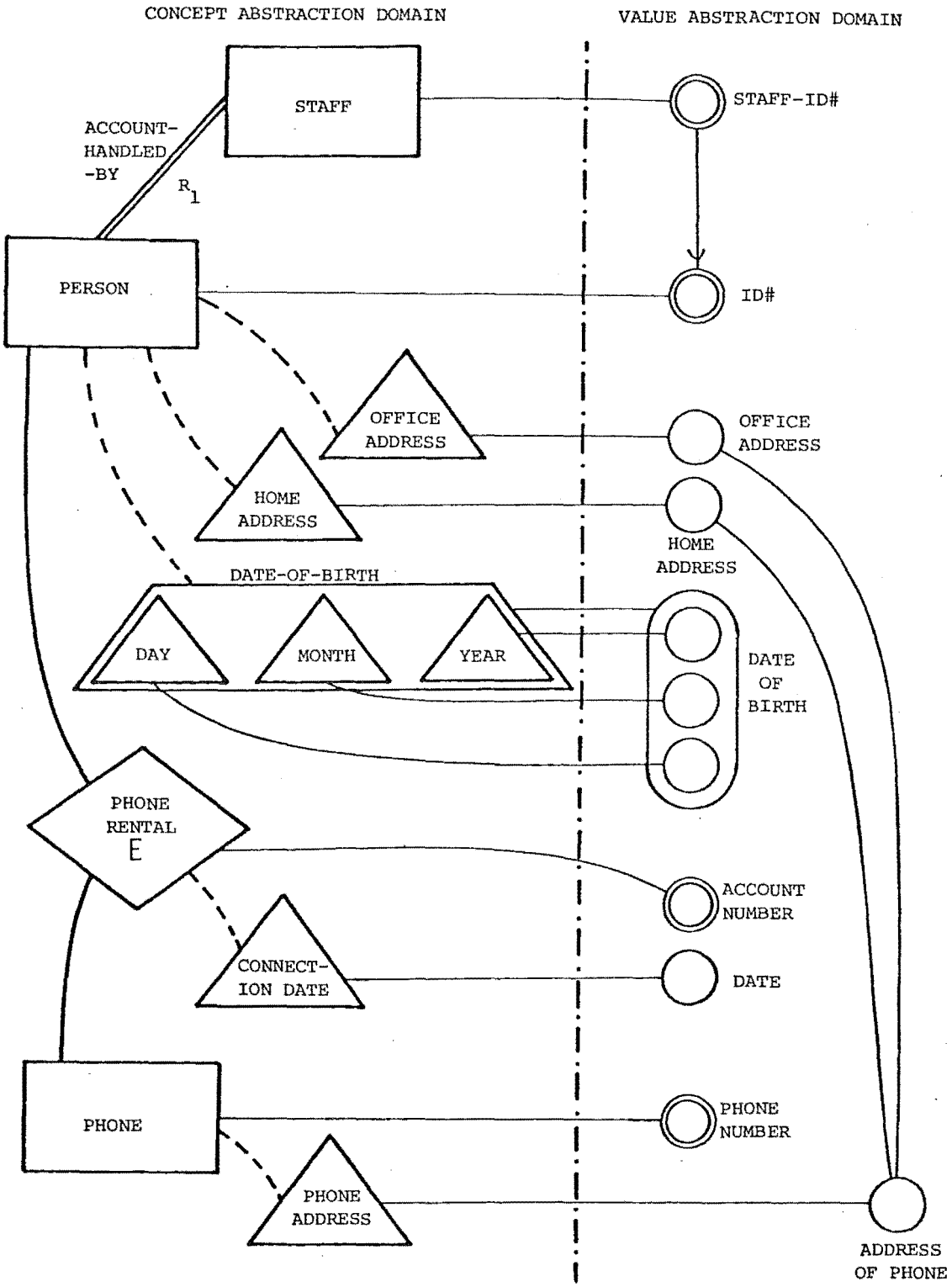


FIGURE 4.1 Example of DATAM diagram

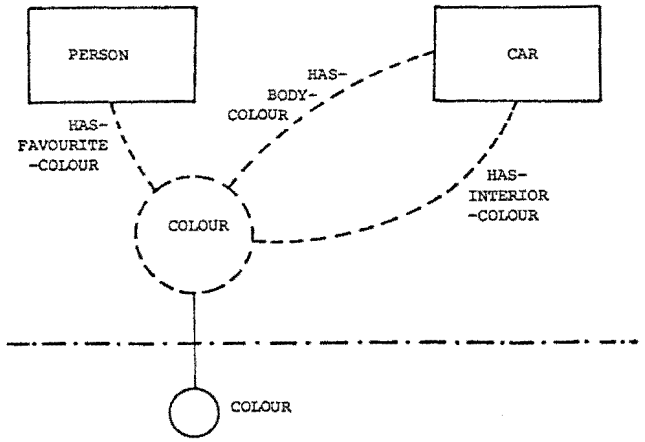


FIGURE 4.2 Multiple attribute associations

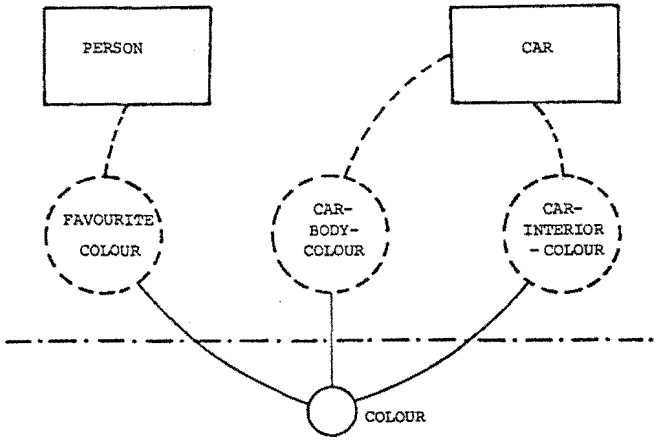


FIGURE 4.3 Factorization of attributes in Figure 4.2

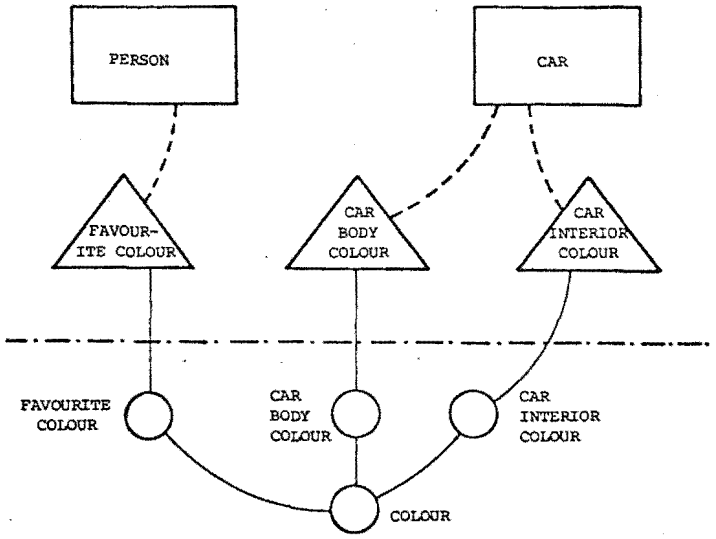


FIGURE 4.4 Factored attributes

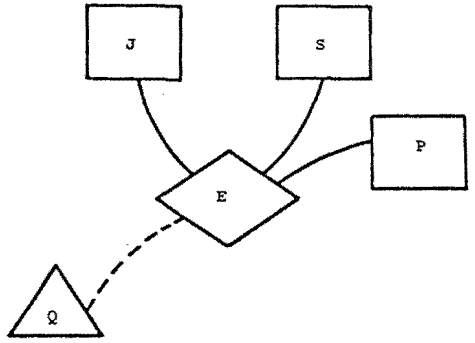


FIGURE 4.5 n-way event

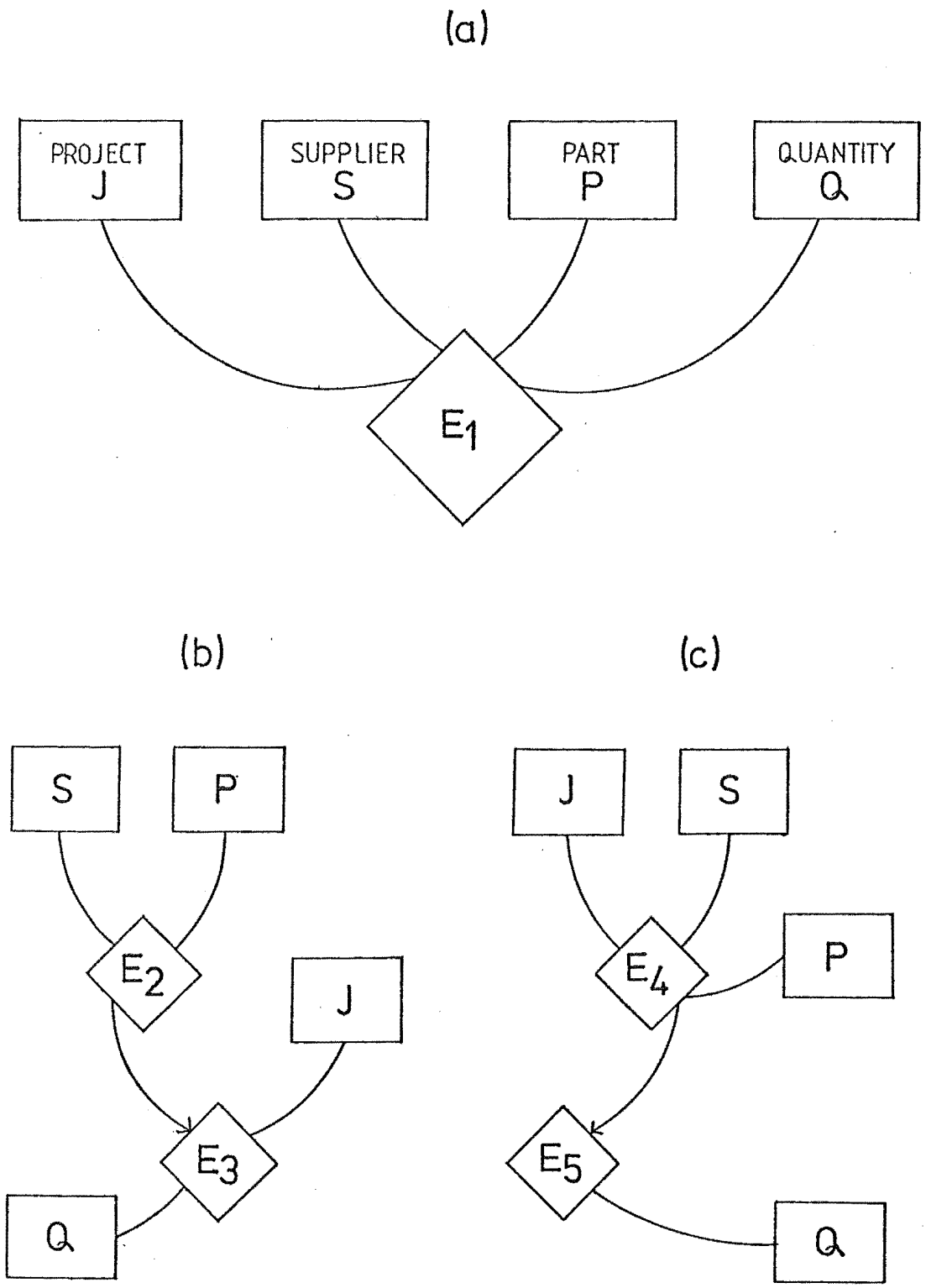


FIGURE 4.6 Entity associations

(a) 4-way event

(b), (c) other representations involving the entities

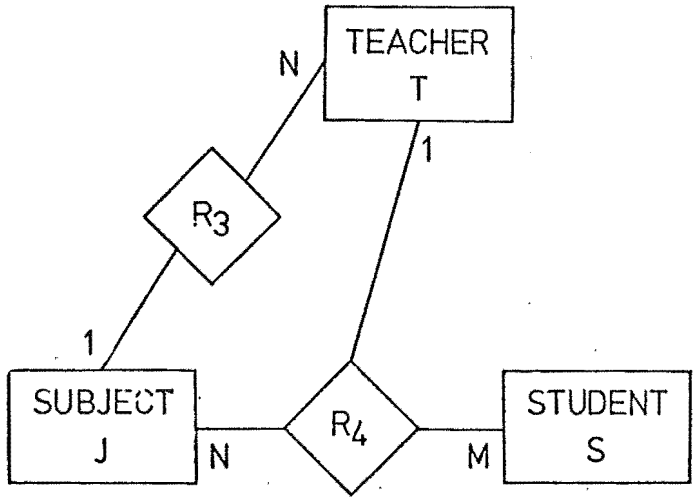
R1 (S , T)

S	T
S1	T1
S2	T2
S2	T3

R2 (T , J)

T	J
T1	J1
T2	J1
T3	J2

FIGURE 4.7 Instances for the relational example in Sec. 4.3.1



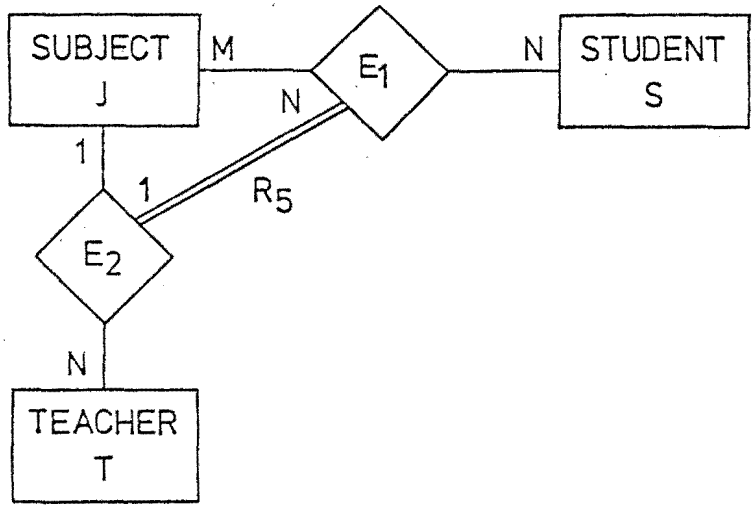
R3 (J , T)

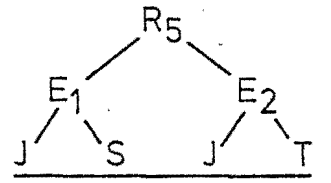
J	T
J1	T1
J1	T2
J2	T3

R4 (J , S , T)

J	S	T
J1	S1	T1
J1	S2	T2
J2	S2	T3
J2	S3	T1

FIGURE 4.8 An entity-relationship model and instances for the example in Sec. 4.3.1





J1	S1	J1	T1
J1	S2	J1	T2
J2	S2	J2	T3
J2	S3	J2	T4

FIGURE 4.9 A DATAM model and instances for the example in Sec. 4.3.1

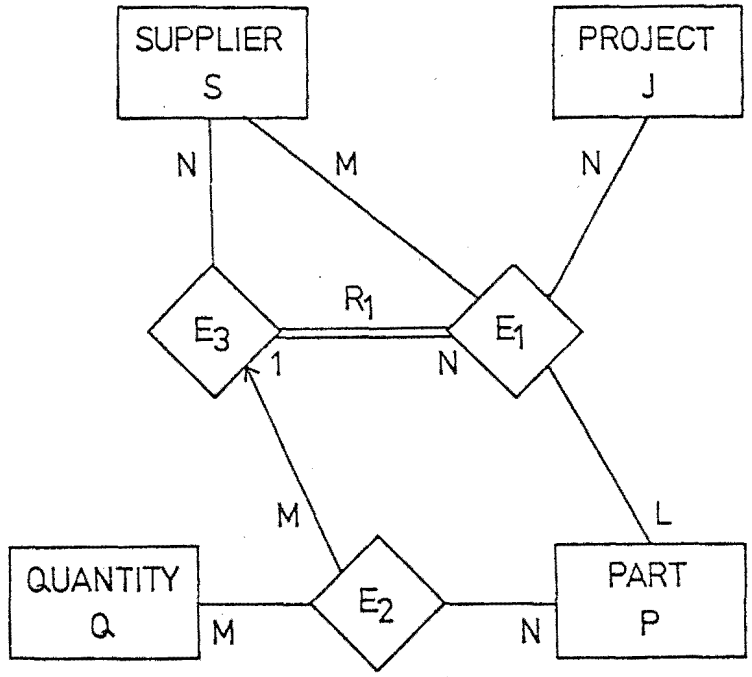


FIGURE 4.10 A DATAM diagram for the supply model of Sec. 4.3.2

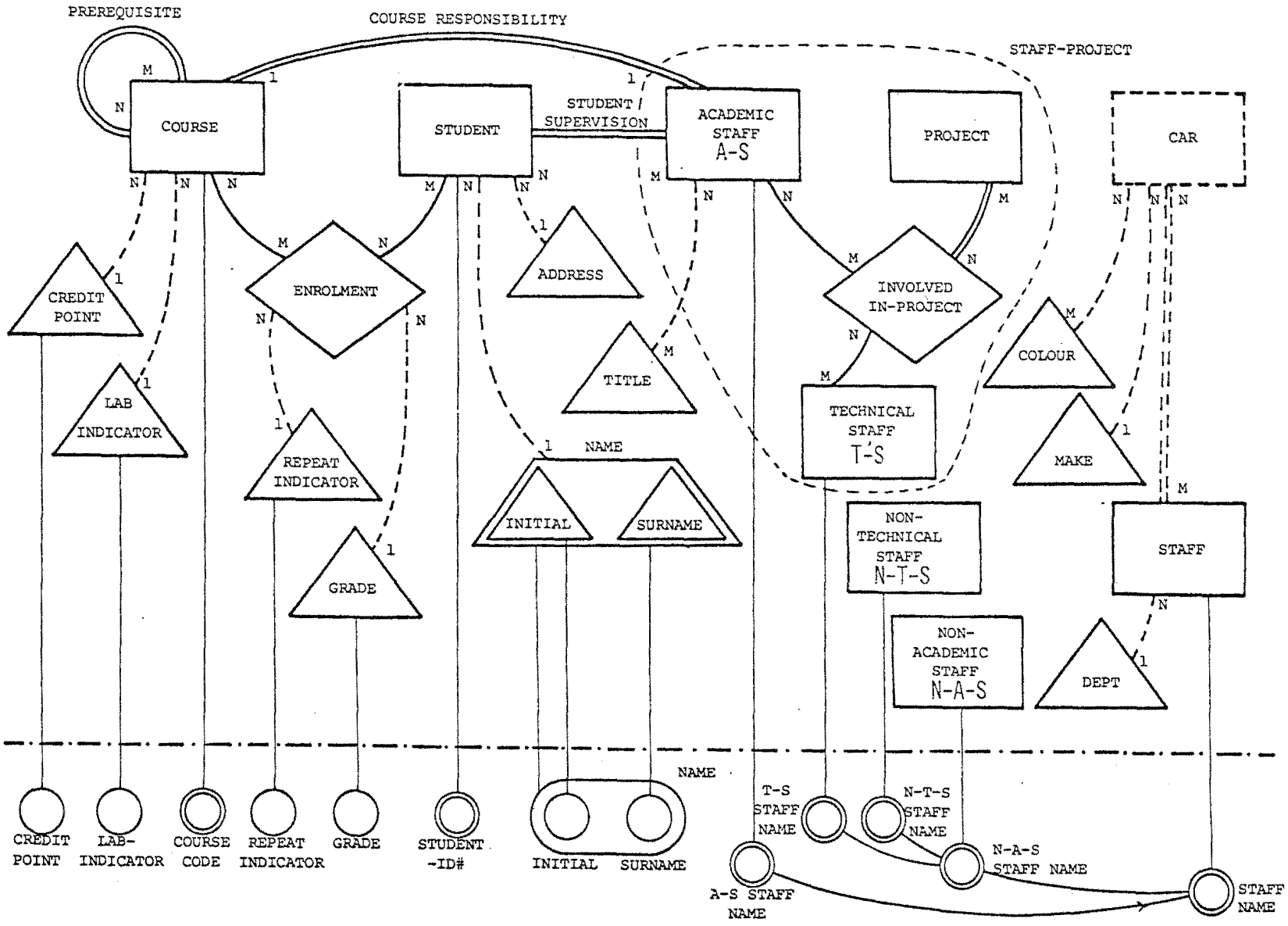


FIGURE 4.11 DATAM diagram for transformation

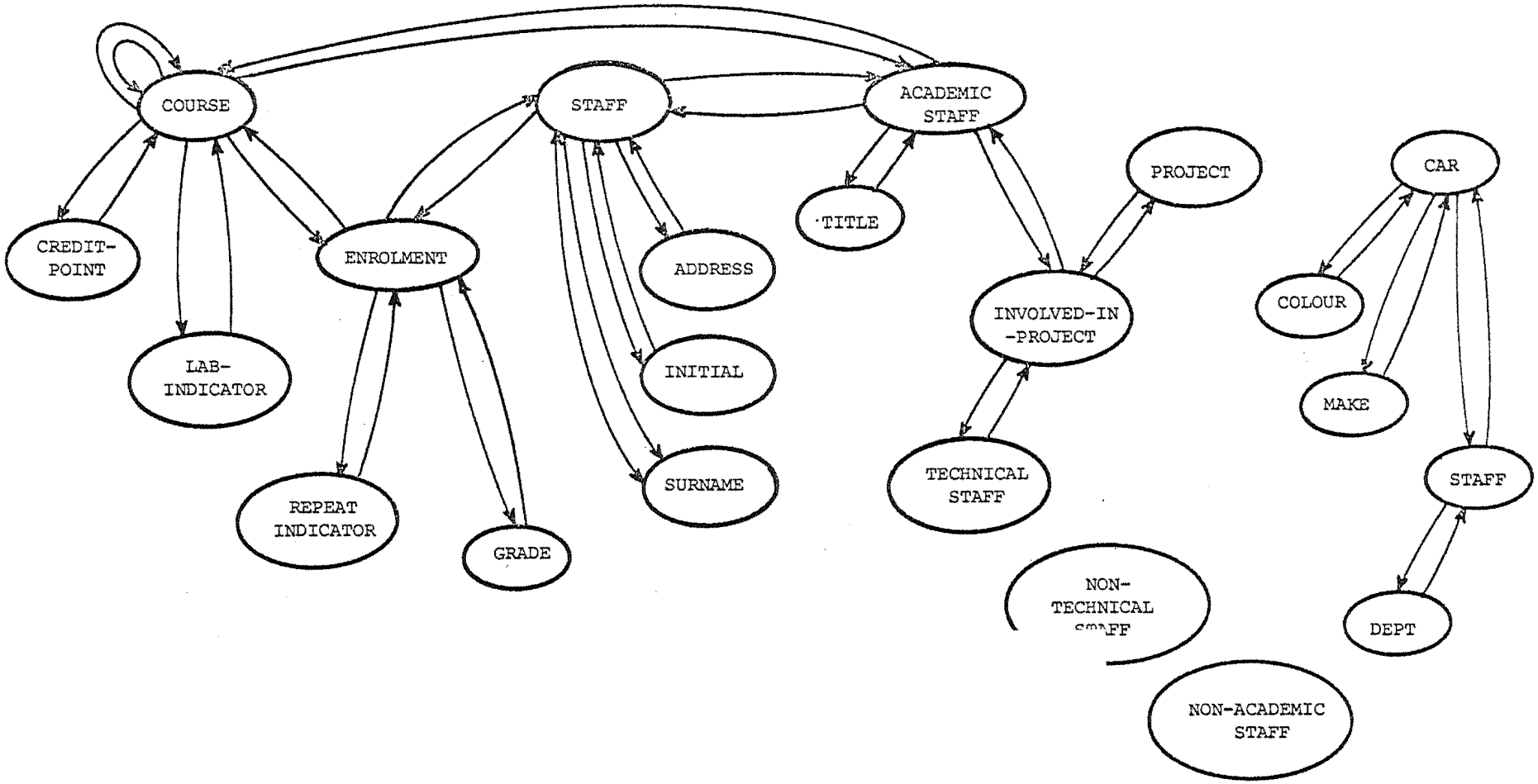


FIGURE 4.12 Binary relational diagram corresponding to Figure 4.11

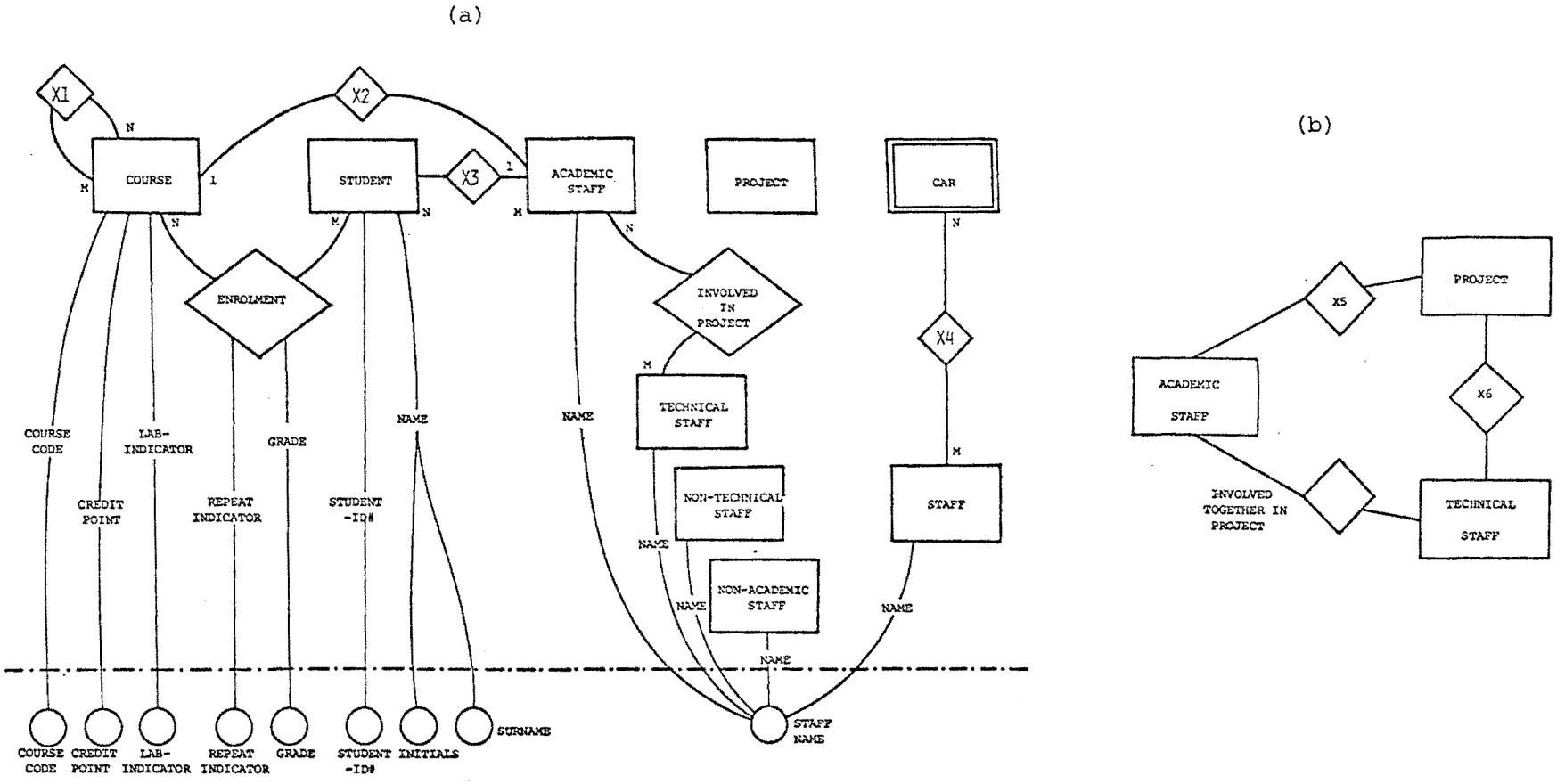


FIGURE 4.13 Correspondence to the ER-model

(a) ER-diagram of Figure 4.11
(b) Alternative substructure

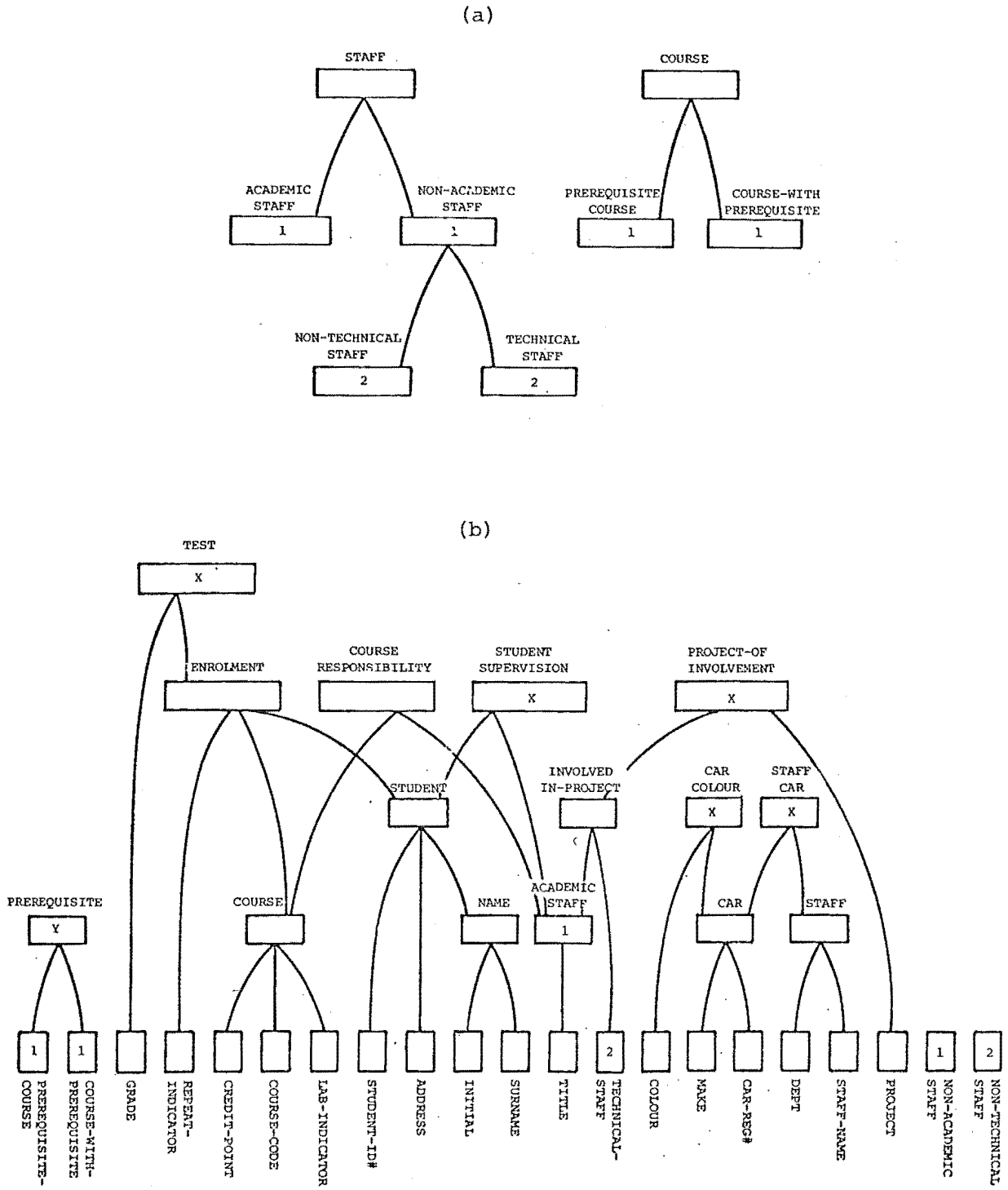


FIGURE 4.14 Correspondence to the aggregation/generalisation model

(a) Generalisation hierarchies

(b) Aggregation hierarchies

I. Progression and regression

(a) Entry and exit of

1. object types: entity, attribute, relationship, event, range
2. object instances: entity, attribute, relationship, event

(b) Factorization and consolidation of object types:

1. independent objects
2. dependent objects based on associational rules
3. dependent objects not based on associational rules
4. composition of instances
5. n-way events

(c) Transformation of abstraction type:

1. attribute ↔ entity
2. entity ↔ event
3. relationship ↔ event
4. dependent entity ↔ regular entity

II. Modification

Change of:

1. range attribute
2. object name: types, instances
3. cardinality of mapping
4. order of instance composition

FIGURE 4.15 Evolution operations

(a)

(b)

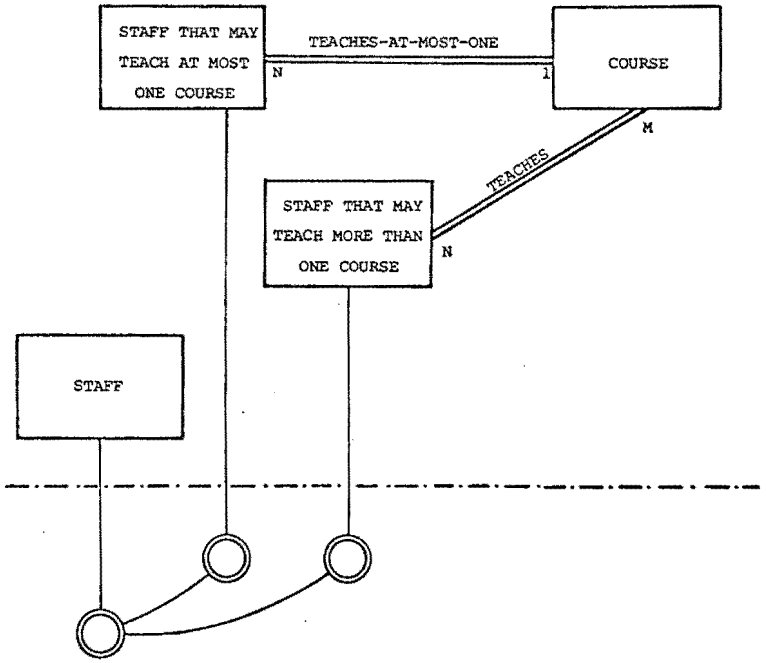


FIGURE 4.16 Sub-objects for varying relationship cardinalities.
(a) Initial configuration
(b) Construction of separate relationship and sub-objects for the different cardinalities

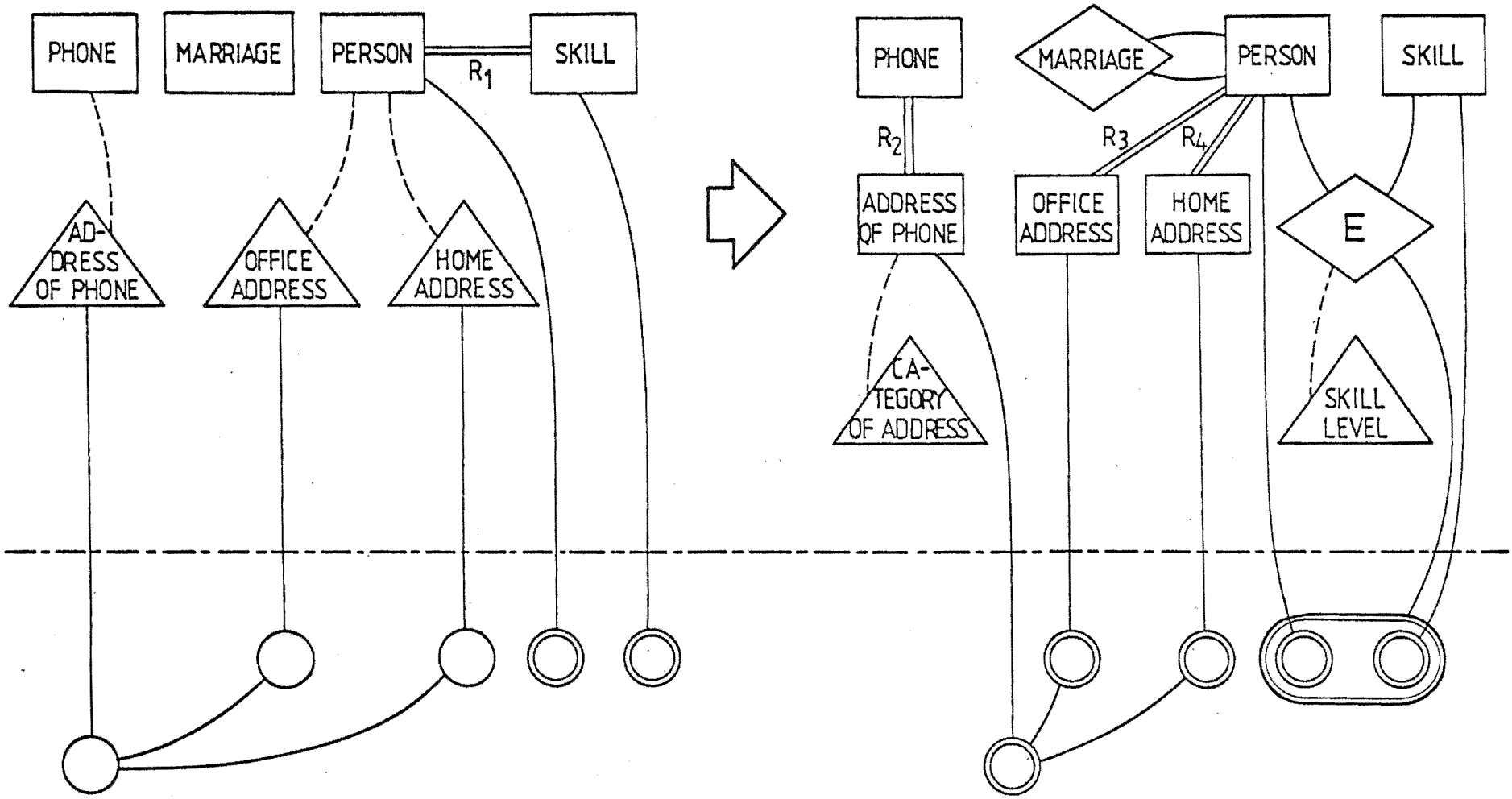


FIGURE 4.17 Evolution of a DATAM model

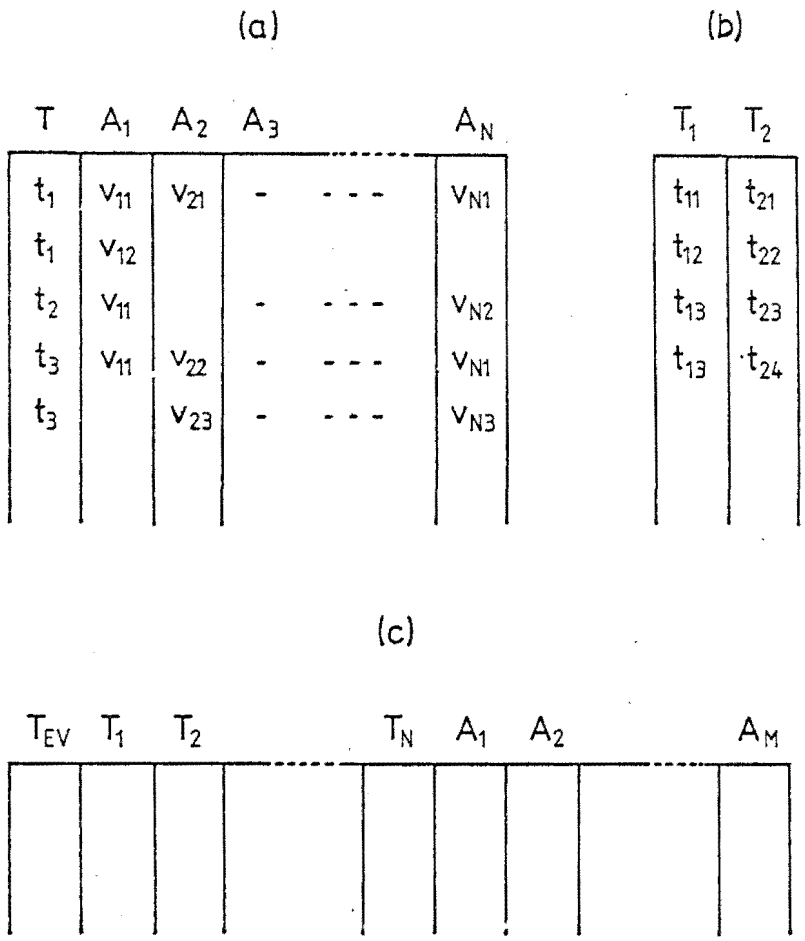


FIGURE 4.18 DATAM instance representations

(a) entity-attribute, EA-table type
(b) relationship, R-table type
(c) event, Ev-table type

(a)

S: {STOCK#, DEPT#} → QUANTITY
T: STOCK# → PRICE
U: DEPT# → CITY
W: CITY → POPULATION

(b)

S (STOCK#, DEPT#, QUANTITY)
T (STOCK#, PRICE)
U (DEPT#, CITY)
W (CITY, POPULATION)

FIGURE 4.19 A relational model for analysis

(a) relational dependencies
(b) normalised relations

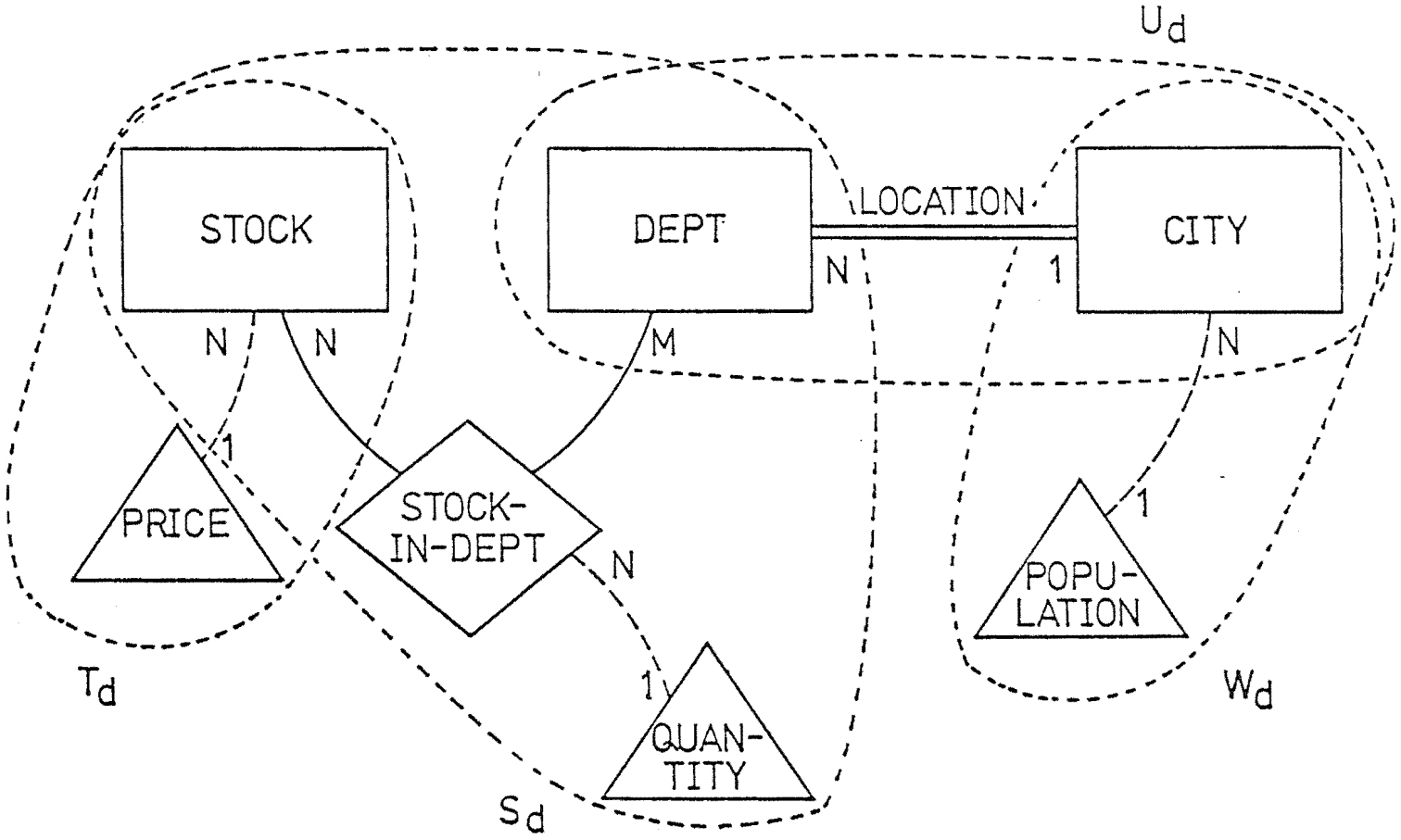


FIGURE 4.20 A DATAM interpretation

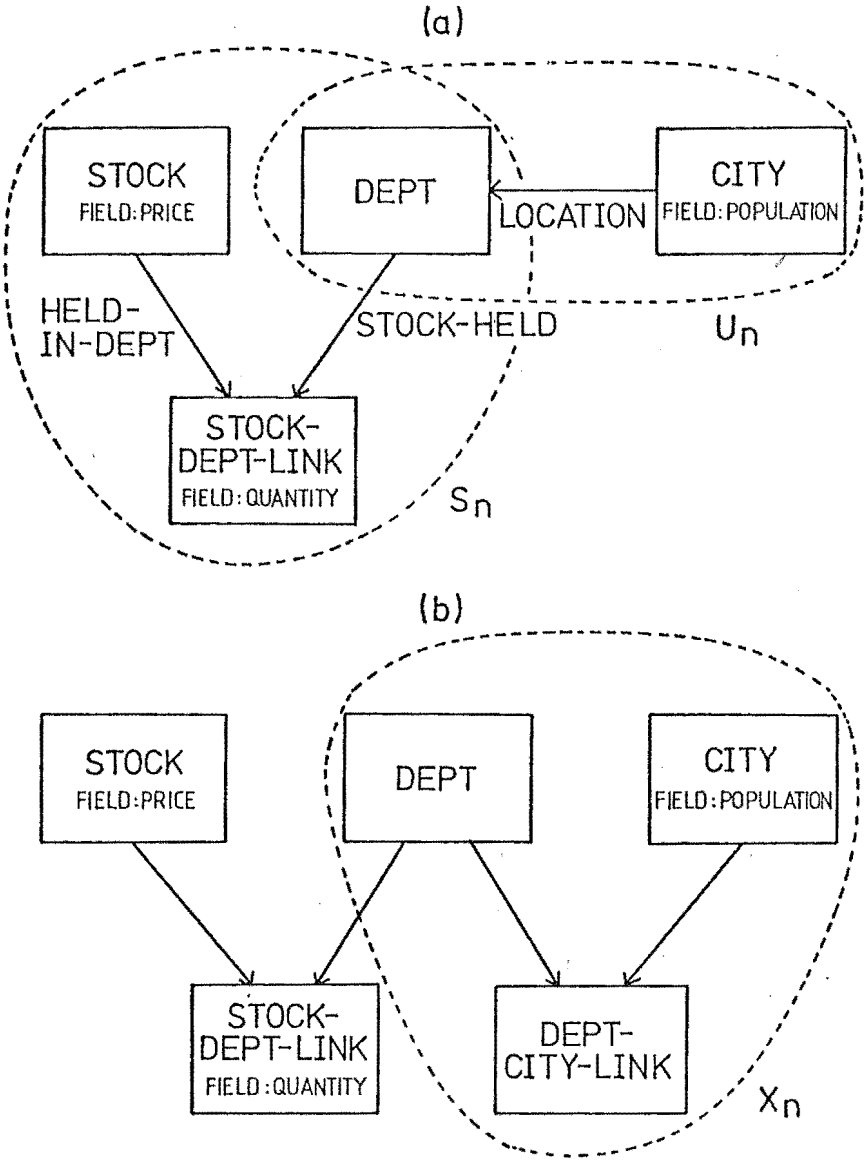


FIGURE 4.21 A network model for analysis

(a) data structure diagram
(b) modified diagram

S: {STOCK#, DEPT#} → QUANTITY

T: STOCK# → PRICE

X: {DEPT#, CITY} → ∅

W: CITY → POPULATION

FIGURE 4.22 Dependencies corresponding to network model

CHAPTER 5

PRIMDAS - A PRIMITIVE DATA STRUCTURE INTERFACE FOR GENERALIZED MODEL SUPPORT

5.1 INTRODUCTION

Criteria for an implementation base include simplicity and flexibility. Flexibility is required in the support of different representations as well as in the support of different interfaces to these representations.

As discussed in Chapter 2, multiple model support at the data structure level in the hierarchy of data representation levels (Fig. 2.1) provides a good efficiency/ease of construction tradeoff. Further, a low level (procedural) approach in conjunction with a set of primitive structural building blocks is seen to provide a good degree of flexibility. Considerations of the actual form and features of a generalized data structure interface are discussed in Sec. 5.2.

Based on these discussions a particular primitive data structure approach, PRIMDAS, is presented in Sec. 5.3. PRIMDAS is proposed as the basis of an implementation interface for the support of a wide range of data base models. The objects and operators of PRIMDAS, which emphasizes simplicity, are described in detail in Sec. 5.3.

PRIMDAS is a generalization which extends data structuring from record-based to item-based considerations. It is an alternative to structuring facilities which consider the support of specific ranges of approaches.

5.2 CONSIDERATIONS OF A PRIMITIVE DATA STRUCTURE INTERFACE

5.2.1 Introduction

The data representation at the data structure level represents and mechanizes the data representation and operations at the conceptual level. As such, the form of the data objects and operators provided at a generalized implementation interface must take into account the relationships of different semantic approaches to their data structure support. This would provide the basis in the design of the implementation base, on which the support of any particular model should not be unduly restricted.

Considerations of the interface are separated into three areas: the influence of conceptual data associations, the form that the data sublanguage should take and the representation of consistency. They are discussed in Secs 5.2.2, 5.2.3 and 5.2.4 respectively.

5.2.2. Influence of Conceptual Data Associations

5.2.2.1 Data associations

The objects at the conceptual level and their associations, which model the real world, will in the final analysis be represented by physical data on which the associations are imposed. At the data structure level, the corresponding objects are the structures in which data is deemed to reside (data pools) and the structures which relate the data pools to each other (see also Tsichritzis [1975]). Each instance or atomic object and their associations at the conceptual level have to be represented at the data structure level, so that operations on and between conceptual data are implemented by corresponding operations on and between data pools.

To reflect the possible operations at the conceptual level, the facilities that are necessary at the data structure level are those to:

(1) define/create/destroy the structural objects

(2) enter/delete/change/retrieve data in the structures.

The design of the actual form of the data objects and operators to be provided requires a set of premises concerning the data base models that the interface is meant to support. Besides providing the basis for design decisions, these premises also serve to determine the applicability of the interface. Because of the disparity and lack of commonality in data base approaches, it is difficult to formulate premises which are guaranteed to encompass all approaches. However, a large class of approaches can be catered for. In this chapter the general conceptual framework of Chapter 3, which encompasses a large number of approaches, is used as a guideline.

Various classifications of approaches in conceptual data modelling have been based on their "structural" differences. Two examples are the predicate calculus/network (Wong and Mylopoulos [1977]) and the graph/set theoretic (Kerschberg *et al.* [1976]) classifications. In all approaches, however, the representation of real world concepts has invariably been categorized into types, representing sets of similarly related conceptual data instances. The entry or fitting of any particular real world fact is then expressed as an instance of these semantic types, as are the retrieval of individual facts from the data base.

5.2.2.2 Associational structures

The implication of the emphasis on type is that the underlying data structures need not vary considerably. Semantic types, whether predicates or association types, normally involve a known number of object types, where a fact or instance of a predicate or association is seen as consisting of instances from each of these objects. An immediate perception is to view instances of each object type as being

in a set, and a fact as an association between items of these sets.

A particular manifestation of this is in the table representations commonly adopted (e.g. relations, record-types), where columns contain instances of object types while rows indicate the association among the data instances to represent a fact. Of these, the relational model is the only one adopting a pure table representation in which no other structures exist and where all data associations are visible in the tables.

However, strictly table-type data structures, as opposed to data perception, do not reflect well typical access actions. In particular, in a pure table representation, associations between tables (in relating different facts) are implicit and involve significant redundancy. Access between tables is better represented by replacing some tables or parts of tables by linking structures (Tsichritzis [1975]), in acting as alternative association indicators. This avoids any ambiguity in the specification of associations as well as reducing efforts in maintaining consistency. Since these considerations apply to the representation of data base models in general, structural associational indicators should be incorporated into a generalized implementation base.

5.2.2.3 Inverted table structure

In access, typically small units of data are required, for which traditional table representations provide row by row access. This means, however, that access of a single item from a row requires access of the whole row. Access of a particular column (e.g. Boyce *et al.* [1974]) may require access of the whole table. This is avoided by representing each set of similar data instances separately. Such an *inverted table* approach is seen to be the more suitable representation for item-based access and for a generalized implementation base, where flexibility is important.

In summary, this section argues that for a generalized

implementation base, the data structuring should be based on single instances, and include structural rather than implicit representations of data associations. Considerations of access on such a structure are discussed below.

5.2.3 Form of the Data Sublanguage

5.2.3.1 Procedural approach

Data sublanguages can take different forms, where each form is seen to have advantages over others. Discussions on the relative merits of particular approaches abound (e.g. Date and Codd [1974], Bachman [1973]). However, as discussed in Chapter 2, differences in the form of the languages are more a function of the differences in procedurality of approach rather than of the differences in representational structures (Held and Stonebraker [1975]). Date [1976], for example, describes an extended data sublanguage catering for three different structural representations.

In the design of the implementation interface, the level of procedurality of the data sublanguage is an important consideration. A high-level approach presupposes a bias and decreases the flexibility of the interface. Since the data sublanguages at a higher data representation level are defined on the sublanguage at the data structure level, a procedural approach at this latter level is necessary to allow more scope in the choice of interfaces at the higher level. For example, a high level data structure interface will make difficult the construction of a procedural conceptual level interface.

The rest of this section is concerned with low level procedural operations that are inherent, but sometimes disguised or hidden in high level access interfaces. The operations are described in terms of an inverted data representation, which is advocated in Sec. 5.2.2.

The manipulation operations in the entering, modification,

deletion and retrieval of data, all have a common operation in that each can be expressed in terms of an *access* of the data structure followed by the appropriate operation. That is, it is useful to view the actions of getting to a point in the structure (without any inference as to how) as an access, while the actual operation of, e.g. retrieval, refers to the process of materializing the target data.

With this view, it is seen that the bulk of data sublanguage operations concerns access. In a low level interface, the specification of basic access operations and the form they take are major considerations.

Access operations can be grouped into two classes: those concerned with the subsetting of data and those concerned with the traversal of the structure. Additionally, it is useful to consider further basic operations which allow the examination of *neighbouring* nodes. These three classes of operations are discussed in Secs 5.2.3.2, 5.2.3.3 and 5.2.3.4 respectively.

5.2.3.2 Marking

Some form of subsetting facility is inherent in the access operations available in data sublanguages. In high level languages, the processes involved in access are hidden. Selection criteria may be stated in a form giving the impression that no intermediate steps are involved in obtaining the target subset. The mechanization of any multiple criteria selection, however, requires some form of internal subsetting sequences.

Additionally, even in high level approaches, subsetting is seen as an explicit process. For example, in the processing environment of an application program, retrieved target subsets may represent intermediate data. If the processing only requires objects one at a time from the target subset, some form of intermediate structure is required to hold the data until it is required, and of which the user has to be aware of.

Also, it has been noted that most users have difficulties with complex query specifications (Chamberlin *et al.* [1976], Deheneffe and Hennebert [1976], Reisner *et al.* [1975]). This is avoided in languages which allow simple queries. These typically involve breaking queries down into sequences of data base subsetting, thus requiring the user to control intermediate structures (e.g. Deheneffe and Hennebert [1976]).

As a final example of embedded subsetting, note that data manipulation operators are sometimes defined explicitly as subsetting operators in that they are defined over data objects and result in another data object (e.g. relational operators (Codd [1970]), CONVERT (Housel and Shu [1976])).

These high level concepts of subsetting require that the underlying system be capable of maintaining some form of intermediate structures for subsetting purposes. This low-level subsetting need not be done by duplication. It is sufficient to have a mechanism which enables subsets to be available as references to the actual objects in the source set. A facility for remembering or to point to subsets of data is called a *marking* facility. A requirement of any marking scheme is that the marked subset retain any associations it originally had. This means that any structural traversals available from a point in the data base should be possible from that point seen as a member of a marked subset.

Facilities for marking need not be low level. *Views* (Astrahan *et al.* [1976]) and *selectors* (Tsichritzis [1976]) are examples of high level marking. Other marking facilities include *Marks* (Brodie *et al.* [1975]) and *Filters* (Bjorner *et al.* [1973]). However, these facilities are based on particular data base models and therefore on particular structures. Furthermore, the interfaces are not sufficiently low level since biases toward particular forms of expression have been introduced.

In the very low level approach of an implementation interface,

marking is seen as explicit operations on single objects. The marking objects are seen as purely structural objects on which explicit control is provided to mark, demark and traverse with respect to single items.

5.2.3.3 Navigation

Navigation is the user-controlled traversal of a data base. A *navigational interface* is one which provides operators and facilities to navigate a data base.

There is debate on whether users should navigate a data base or leave the traversal to some access engine (Bachman [1973], Codd and Date [1974]). This is not the issue in the design of a primitive implementation interface. Since interfaces providing explicit traversal may be preferred, it should not be precluded by an implementation base. In providing this flexibility, the implementation interface itself should be low level and of a navigational nature.

Given any structure, there are typically various ways of traversing it. To avoid the necessity of choosing, or to reflect particular usage patterns, navigational interfaces are often restricted to selected paths. In an implementation interface, such restrictions are to be avoided. The traversal operators provided must form a complete set with respect to the inherent paths of the structure.

In navigation, the mechanism to keep track of the position of the traversal is called a *cursoring* mechanism. A *cursor* designates an object in the structure as that which the traversal is currently at. This current object can then be operated on: retrieved, updated, and deleted.

The navigational operators consist of setting the cursor to a particular position in the structure. This could be done by moving the cursor relative to its current position or some fixed position in the structure. Additionally, a position can be specified by the *value* of the object at that position.

In complex navigation, it is necessary to maintain more than one position in the data base. Therefore an implementation interface has to provide for *multiple cursoring*.

This section has considered general navigational features. The actual form that the navigational operators take is dependent on the structural objects available.

5.2.3.4 Associational access

In traditional representations, access of a particular item is typically accompanied by mandatory access of some of its related items (e.g. record access). With separate instance set structuring, where single sets of items are accessed, alternative facilities are required to access related items.

This is represented by the *associational access* facility which allows access to items "close" to ones given. Typically the requirement is to operate on these nearby objects, whilst maintaining positions at the reference objects. Rather than employing navigation followed by retracing, a more representative facility would be to allow close objects to be operated on without actually going there. This represents the function of associational access.

The objects that are to be within reach have to be well-defined. Three basic forms of structural association are isolated, corresponding to three forms of associational access: *direct association, indirect association* and *common link association*.

Direct association is used to specify those objects closest to a given one. In a table representation, the entries in a row are viewed as being closely related and would be specified as directly associated to each other. A characteristic of direct association is that it does not involve any intermediary linking structures. This means that there can be at most one direct association between any two objects, whether sets or instances.

Indirect associations specify the association of objects through some linking structure. There may be many such associations between any two objects. Indirect associations are extended to the directly associated objects of the associated objects, so that in a table representation it corresponds to the association of items in a row of one table to those of another table.

The distinction between direct and indirect associations is evident in query languages (e.g. Senko [1976a], Deheneffe and Hennebert [1976]), where different query structures are available for the case where the item required is directly available and the case where implied data associations need to be resolved.

In both direct and indirect association, the association is defined between single objects. In common link associations, a set of two or more objects (source objects) are jointly associated to one object (target object). The association of the source objects to the target object is represented through a set of linking structures.

As with indirect associations, common link associations are extended to directly associated objects. The corresponding configuration in table structures is where two or more tables together, are associated to another table. However, the typical view in table approaches is that of each component association existing independently, rather than as part of a unit (cf. set-types in the network model). With this perception, the joint nature of the association is lost.

Common link associations are perceived also to embed the indirect association corresponding to each component association.

5.2.4 Representation of Consistency

Constructs of abstract models embody formalisations of data consistency to represent real world perceptions. At the data structure implementation base, the consideration is to determine those consistency

features which are general enough to be incorporated into the base. Such features need to be representable as strictly structural concepts with no inherent semantics.

Two such structural consistency concepts are those involved in ensuring that subsets remain subsets and in ensuring the equivalence of alternative associational structures. The former could be used to directly represent IS-A associations, while one of the uses of the latter is in constructing alternative views where different structures may be used to represent the same concept.

These structural consistency features are generally applicable, and their incorporation into an implementation base would increase their data base model support potential while retaining generality.

5.3 PRIMDAS - A PRIMITIVE DATA STRUCTURE INTERFACE

5.3.1 Introduction

This section describes PRIMDAS, a low level structural facility incorporating the general features isolated in Sec. 5.2. PRIMDAS consists of a set of structural building blocks and operations on them. Embedded within a suitable host language, PRIMDAS forms the basis of an abstract machine on which different data base models can be readily implemented.

The structural objects of PRIMDAS are described in Sec. 5.3.2. The description includes the operators with which to construct the objects. Diagrammatic representations of PRIMDAS are given in Fig. 5.1 and will be used to illustrate PRIMDAS configurations.

Access, manipulation and restructuring operators of PRIMDAS are presented in Sec. 5.3.3. Fig. 5.2 lists the operators to be considered.

Finally, Sec. 5.3.4 specifies the form of consistency embedded within PRIMDAS.

5.3.2 PRIMDAS Structural Objects

There are three types of structural objects in PRIMDAS: *value sets, link structures* and *mark sets*. A single operator, CREATE, is used in the construction of these objects. Its general form is

$$\text{CREATE} \left(\left\{ \begin{matrix} \text{VALUESET} \\ \text{LINK} \\ \text{MARKSET} \end{matrix} \right\} , \langle \text{parameter list} \rangle \right)$$

where <parameter list> are the arguments required for the object type being constructed. Each of the PRIMDAS object types is described in the following subsections.

5.3.2.1 Value sets

These are the unit structures in which data can be stored, representing the data pools of a data representation interface. All items in any single value set are of the same type. This type is specified when constructing the value set.

Other attributes to be specified are those necessary for its implementation and use. In a typical implementation, the information required would include a unique name and the item length. With these, the value set construction command is of the form

CREATE (VALUESET, <name>, <type>, <length>)

Each value set is seen as a linear list. Navigation on such a set would require some form of ordering, which in the general case may be user specified. The order could be chosen to be that most suitable for its use.

Associations among value sets include that representing direct association types. The operator to specify direct association is

DIRASS (<value set list>) .

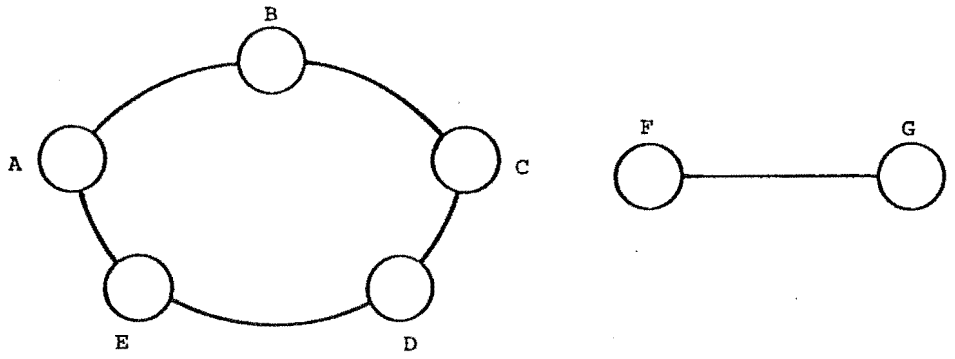
This directly associates the value sets in the list to each other. The value sets in this list may already be directly associated to other

value sets. In this case the result is the union of all the value sets, forming a mutually associated set. For example, the sequence of operations

```
DIRASS (A,B,C)
DIRASS (B,D,E)
DIRASS (F,G)
```

would result in the associated sets {A,B,C,D,E}, {F,G}.

A diagrammatic representation of this configuration is given below.



Where there are more than two value sets, the association is depicted as a ring to emphasize the mutual association among the value sets.

This association among value sets provides a structural configuration within which associations among their items can be specified. A diagrammatic representation is required to illustrate items in individual value sets as well as the association of items between value sets. The representation should emphasize both the independence of items in a value set from those in others as well as their associations to items in other value sets. An example of the representation adopted here is given in Fig. 5.3, where each value set is represented by a two column table. The right-hand column consists of the items in the value set in the order it is perceived to be stored. The left-hand column contains a unique index for the corresponding item in the other column.

Direct association of items from different value sets are indicated by the items having the same index. For example, consider

the instances of Fig. 5.3, where A, B and C are directly associated value sets. a1 is associated to b2 and c4, a2 is associated to b3, a3 is associated to c2, while a4 is not associated to any item of B or C.

5.3.2.2 Link structures

Indirect associations are recorded into explicit intermediary structures, called LINKs. A link structure is specified between two value sets or mark sets (Sec. 5.3.2.3). The link is said to be *incident* to each of the two structures. The CREATE command for links has the form

CREATE (LINK,<link structure name>,<value/mark set 1><value/mark set 2>)

In most situations, the direction of the link is of no consequence, since this is clear from the context in which the link is used. However, where ambiguous situations occur, for example when set 1 is the same as set 2, the direction in which a link is to be used has to be indicated. Here, the positive direction is arbitrarily taken to be from set 1 to set 2.

The diagrammatic representation of linked value set configurations is illustrated in Fig. 5.4. The indirect associations of value set A to value sets B and C are indicated by the lines joining A to B and A to C. The names on the lines are those of the link structures representing each association.

The entries of link structures facilitates the association between items of the associated value sets. Since items in value sets are uniquely identified by an index, an appropriate diagrammatic representation of link structure entries consists of two columns, where each column contains indices of items of each of the value sets associated by the link structure. The rows are the link entries which represent pairs of associated items. A link entry is said to be

incident to each of the items it associates.

An example is given in Fig. 5.4(b), illustrating the instances and associations of a subset of the configuration in Fig. 5.4(a). The columns of the L1 link entries are labelled to indicate which value set they are indices of. From these entries it can be seen that the item a1 is associated through L1 to the items b1 and b2, items a2 and a3 are both associated to b3, while a4 is not associated to any item of B. Additionally, indirect associations extend to directly associated value sets. This means that the link structure L1 in effect also associates items of A to items of D. That is, a1 is associated to d4, while both a2 and a3 are associated to d1.

A value or mark set may participate in many links.

Common link associations, which are also represented with link structures, require that a value/mark set be common to more than one link. In Fig. 5.4(a), the configuration involving A, B, C, L1 and L2 could be used as a common link association. The entries of A could be such that they are determined by B and C through L1 and L2.

5.3.2.3 Mark sets

One form of subsetting is to keep track of the marks of, or pointers to, the actual values. These marks are stored in *mark sets*. A mark set is defined on a value set or another mark set. It is constructed with the operator

CREATE (MARKSET,<mark set name><source set>)

The values only exist in the value set on which the marks are defined. The existence of marked items is determined by their existence in the value set. This is particularly useful in situations where the consistency of subsets is to be maintained.

To reflect its usage in subsetting, mark sets are traversable and behave, for access, like their value set. The difference is that only marked items are available.

Other than in temporary subsetting for the purpose of access, mark sets may also be used for permanent subsetting. Such a mark set may represent, for example, the instance set of a sub-entity (Chapter 6).

To allow the update of values with respect to the subset, update operations available on value sets are also available on mark sets.

Directly associable value sets may be attached to mark sets. This means that it would be possible, for example, to represent the situation where the data associations of sub-objects are to develop independently of those of the source object (Chapter 6).

More than one mark set may be defined on a source value/mark (v/m) set.

An example of a PRIMDAS configuration involving mark sets is illustrated in Fig. 5.5. The association of a mark set to its source set is indicated by the dotted arc. In the figure, two mark sets MA1 and MA2 are defined on A. MA1 itself has a mark set, MA11. Also, the mark set MA2 is independently associated to X through the link LM.

The diagrammatic representation of mark set entries adopted here is identical to that of value sets (Subsec. 5.3.2.1). The marked items are represented with the same indices as the corresponding values in the value sets.

5.3.3 PRIMDAS Operators

5.3.3.1 Introduction

PRIMDAS is a navigational interface on which operators are provided to explicitly traverse the structures. Navigational facilities for traversal within value sets are described in Subsec. 5.3.3.2, while inter-value set navigation is considered in Subsec. 5.3.3.9.

Other than the entry of direct associations, all the other

operators are defined in terms of positions in the data base - i.e. cursors. An implication of this is that indirect associations can be specified only on data already existing in the data base. This reflects the conceptual situation where objects of interest are specified first, before the specification of their associations.

Descriptions of PRIMDAS operators, grouped according to their functions, are given in the following subsections. Associational access operations are described in 5.3.3.3, while the specification of the different forms of associations are given in Subsecs 5.3.3.4 to 5.3.3.6. Data manipulation operators are discussed in 5.3.3.7, while operators involved in subsetting are presented in 5.3.3.8. Finally, operators to perform restructuring of PRIMDAS configurations are considered in Subsec. 5.3.3.10.

Although the operators are described in terms of value sets, they apply as well to mark sets. Where differences occur, they will be made clear in the discussion.

5.3.3.2 Navigational facilities

The navigational units of PRIMDAS are the value sets. Cursors are defined to range only within value sets. That is, cursor setting operators can only move a cursor to items within the value set on which the cursor is defined. The items in each value set are ordered in some way, so that a sufficient set of operators to traverse value sets are the operators

FIRST
NEXT .

However, to allow more flexibility in traversals, the operators

LAST
PRIOR

are also required. To set a cursor to an item whose value is known, the operator

MATCH (<value>)

is used, where the required value is an operand of the operator.

Multiple cursors are available on PRIMDAS in two modes. In one, each cursor is given a unique name, which is specified when creating the cursor by the operator

SEARCH (<value set>,<cursor name>)

To set such a cursor, the cursor setting operators have to explicitly indicate the cursor to be set,

e.g. MATCH (<cursor>,<value>)

Cursors can be destroyed by using the End Search operator,

ENDS

which also has to be qualified by the cursor name when destroying named cursors.

The other way in which multiple cursors are made available is by imposing a stacking order on the use of unnamed cursors. The *current* unnamed cursor is that most recently created. All cursor setting operators not qualified by any cursor name are interpreted to be defined on the current unnamed cursor. An unnamed ENDS causes an unstacking so that the next most recently created unnamed cursor becomes the current one.

The use of unnamed cursors allows routines to be able to contain SEARCHes without the need to ensure that their names are unique.

More than one cursor can be defined on the same value set at any one time.

5.3.3.3 Associational access

The three forms of associational access: direct, indirect and through common links, represent the action of "reaching out" and do not set any cursor.

Source cursors are required to indicate the position from which the access is to be performed. Access is made to an item in a value or mark set associated to the ones in which the source cursors are defined. Both direct and indirect associations require one source cursor, while common link association needs at least two.

Operationally, direct association can be seen as a special case of indirect association. In PRIMDAS, both forms are invoked with the ASSOC operator, while the AT operator caters for common link association. The rest of this section describes first the ASSOC operator and secondly the AT operator. The use of associational access expressions as generalized data base settings is then discussed.

5.3.3.3.1 ASSOC operator

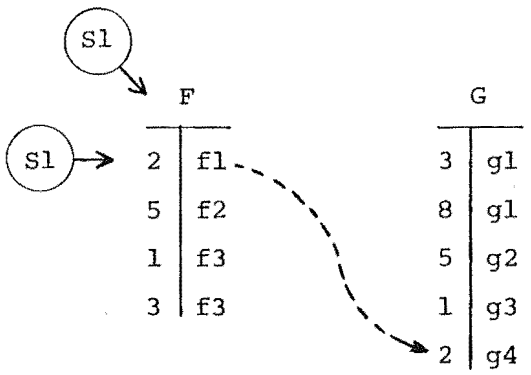
The general form of ASSOC is

ASSOC(<$\ell_1,\ell_2,\ldots,\ell_n$>,<source cursor>,<target v/m set>)

where the ℓ_i are PRIMDAS link structures.

In a direct associational access, link structures are not required, so that n is 0. Following is an example of PRIMDAS code illustrating a direct ASSOC from the value set F to the value set G, where G is directly associated to F. The contents of the value sets and the actions corresponding to the operations are also indicated.

```
1.  SEARCH (F, S1)
2.  FIRST  (S1)
3.  ASSOC  (<>, S1, G)
```



On the values shown, the code will return the value g4.

To illustrate an indirect ASSOC, consider the example in Fig. 5.6

where the value sets X,Y are associated through the link structures L1, L2. The access indicated by the dotted lines represents the code

```
SEARCH (X, S2)
FIRST  (S2)
ASSOC  (<L1, L2>, S2, Y)
```

which obtains the value y1. The link structures in this case have been used in the direction from X to Y. The columns labelled I in both L1 and L2 indicate the intermediate value set to which X and Y are associated to through L1 and L2 respectively.

A reverse access, from Y to X, is indicated by the solid lines. This corresponds to the code:

```
SEARCH (Y, S3)
MATCH  (S3, "y4")
ASSOC  (<L2, L1>, S3, X)
```

getting the value x2.

5.3.3.3.2 AT operator

In common link associations, access of a target item is made in terms of a number of links. These link structures are all incident to the target v/m set, while the item obtained is that to which link entries are commonly incident. The AT operator, which invokes common link associations, has the form

$$\text{AT } (\langle\langle \ell_1, c_1\rangle, \langle \ell_2, c_2\rangle \ \ldots \ \langle \ell_n, c_n\rangle\rangle, \langle\text{target v/m set}\rangle).$$

Each ℓ_i is a link structure incident to the target v/m set and c_i is a cursor defined on the source v/m set of ℓ_i.

As an illustration, consider the configuration and instances of Fig. 5.7. The items of C can be determined by the items of A and B through the links L1 and L2. For example, from the cursors SA, SB in the figure, the access of an item in C can be invoked by

```
AT (<<L1, SA>, <L2, SB>>, C) .
```

This obtains the item c1 as indicated by the dotted lines.

5.3.3.3.3 Multiple associations

Link structures may contain many associations for any single item or set of items. Since associational access is to return only a single value, a convention is therefore required on which one to pick when accessing through such links. The convention adopted is to choose the chronologically first association. Note that there are no choices in direct associational access, since a v/m set item can be directly associated with, at most, one item in any given value set.

In indirect associations, if the access is through more than one link structure, then the association chosen is the first complete association. This is shown in Fig. 5.6. The access from X to Y through L1, L2 uses the second association of x1 in L1, since the first has no corresponding association in L2.

5.3.3.3.4 Data base setting

In the discussion above, ASSOC and AT are described as operators to mechanize associational retrieval in which an item is obtained. However, seen as expressions, the specifications of ASSOC and AT can also be used as data base position indicators. As *data base settings*, they can be used in operations requiring a position indicator (Sec. 5.3.3.7). To amalgamate the various forms of basic positionings available, a PRIMDAS data base setting is defined to be any of the following:

(1) cursor position
(2) ASSOC setting
(3) AT setting

5.3.3.4 Specification of indirect association

Link structures keep track of the indirect associations between items of pairs of value sets. The specification of associations is done by the LINK operation, which links a pair of items. The particular

structure to be used has to be specified in the operation. Positions of the two items to be linked are given by two cursors.

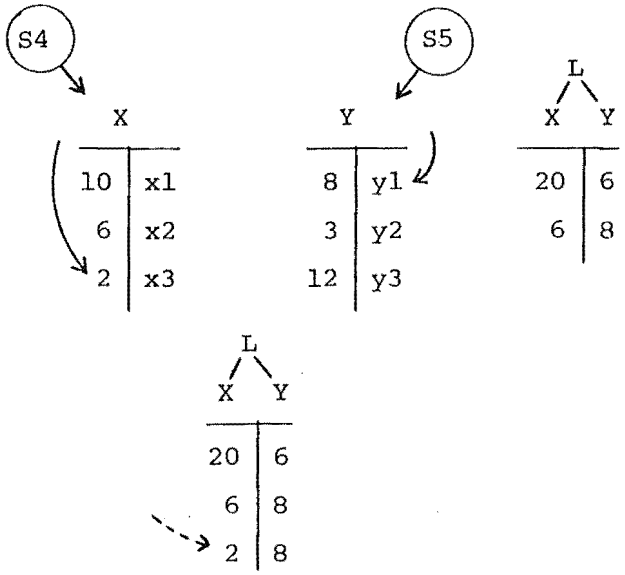
The form of the operator is

LINK (<link structure>,<<cursor 1>,<cursor 2>>) .

Where the direction is ambiguous, the positive direction is taken to be from cursor 1 to cursor 2. Entries in a link structure are in chronological order.

Following is an example, where the association of a pair of items from the value sets X, Y are entered into the link structure L, defined between X and Y.

```
SEARCH   (X, S4)
MATCH    (S4, "x3")
SEARCH   (Y, S5)
FIRST    (S5)
LINK     (L,<S4, S5>)
```

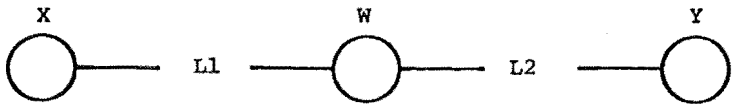


This code would result in an entry into L of the form:

A particular link between two items is destroyed with the operator

DELINK (<link structure>,<<cursor 1>,<cursor 2>>) .

In the case where two value sets are associated through more than one link structure, there necessarily exist intermediate value sets such that the association can be expressed as a sequence of associations through single link structures. For example, consider the case where value set X is associated to value set Y through the link structures L1, L2 in that order. Then there exists a value set W such that



This means that the association of items of X and Y is specified as pairs of associations of X, W items and W, Y items.

5.3.3.5 Specification of direct association

While the specification of indirect association is provided for by operators on data existing in the data base, the specification of direct association between items is done on the entry of the data.

Two operators are available. One, called ENTER, is for the entry of directly associated data which are isolated and independent of other data in the data base. The other, APPEND, allows input data to be appended to directly associated data already existing in the data base.

The ENTER operator has the form:

ENTER (<list of value sets>,<list of cursors>,<source>) .

The operator assumes that the items are entered in sets of directly associated items, each set consisting of one item from each of the given value sets. The value sets are required to be directly associated. The <source> argument indicates the source and mode in which the data is made available. Data is expected to be entered as character strings, with appropriate conventions for item and set delimiters. To allow transformation of available data to the desired form and to perform other operations on the input data, <source> may be a procedure, either user-defined or system provided.

Each of the cursors in the optional parameter, <list of cursors>, is defined on one of the input value sets. Each cursor is set to the most recent item input in that value set. This is useful in input-procedures, particularly where individual sets of associated data are

to be operated on as they enter the data base.

The form of the APPEND operator is

APPEND (<value sets of items to be duplicated>,<input value sets>,<source cursor>,<list of cursors>,<source>).

Here, since items are to be attached to existing data, a cursor is needed to indicate the particular item on which to append. This cursor is the <source cursor> in the command above. As with the ENTER command, items can be input with a number of value sets, as indicated by the parameter <input value sets>. The arguments <list of cursors> and <source> are as for ENTER.

An item in a value set can be directly associated to only one item in any other particular value set. This means that problems arise, where in any one of the input value sets, there already exists an item directly associated to that being appended to. Such an attempt to associate one item to more than one in another value set is resolved here by duplicating the item to be appended and attaching the input data to this duplicate. At times, however, it is useful in such a situation to duplicate not just the one item, but other existing ones which are directly associated to it. The items to be duplicated are specified by listing their value sets in the parameter <value sets of items to be duplicated>. This list and the input list have to be disjoint.

5.3.3.6 Specification of common link association

In common link associations, a number of links are incident to a common value set. This is reflected as well in the instances, where entries of the links are incident to common instances of the value sets. To maintain this form of association, further instances of the common value set would also be specified in terms of the common links.

Consider the configuration and instances of Fig. 5.7. The items of C are such that for each c_i, there exists a_j and b_k which are associated to it through the links L1 and L2 respectively.

As described in Subsec. 5.3.3.3, items in C, for example c1, can be specified by a link setting of the form

<<L1,SA>,<L2,SB>> on C.

In conjunction with APPEND such a link setting specifies the entry of further items into C. The entry is seen as an append, since it is in terms of existing data. The APPEND extends to directly associated value sets of the common value set. For instance, an item for D, say "d4" can be appended to c1 by the command

APPEND (<C>,<D>,<<L1,SA>,<L2,SB>>,<>,<source>)

In such an append, if any values need to be duplicated in C, then corresponding links are entered into L1 and L2. The entries are required to enable the new C item to be determined through the two link structures. For example, the append of another item, "d5", to c1 is illustrated in Fig. 5.8. The arrows indicate the items entered into each of the structures (<nid> is a new unique index). Where the append is directly into C, a new item is always created, so that appropriate entries are always required for the links.

5.3.3.7 Data manipulation

There are three operators that manipulate data. These are GET, DELETE, and REPLACE. Each of these requires a data base setting to indicate the item on which the operation is to be carried out. The general form of these operators is

OPERATOR (<data base setting>,<list of operator specific parameters>).

GET is a generalized retrieval operator, which makes the item at a data base setting available to a user program. Although the exact form in which the data is made available is specific to the implementation, it should be in a program usable form.

The operator, DELETE, enables individual data items to be

removed from the data base. It requires no operand other than the data base setting. If the item to be deleted is that at which the cursor is currently at, the desired effect on the cursor as a consequence of the action has to be predefined. In PRIMDAS, this situation is resolved by moving the cursor to the following item in the value set if one exists, otherwise to the end of set marker.

Furthermore, in a multicursor environment, such as in PRIMDAS, rules have to be formulated in cases where update is to be performed on items pointed to by more than one cursor. Since hidden consequential action on cursors could lead to uncontrolled situations, update of items on which multiple cursors are defined are not allowed. Since items may be specified by associational access, this restriction is extended, so that an item may be updated only if the data base setting that indicates the item to be updated is the only one defined on that item at that time. This represents an item level resolution of concurrency.

There are other considerations in concurrency, particularly where hierarchies of mark sets (Subsec. 5.3.4.1) are involved. However, concurrency is not treated rigorously in this thesis and these further forms of concurrency are not discussed.

The above considerations apply in the use of the REPLACE operator. This operator, which has the form

REPLACE (<data base setting>,<replacement value>),

replaces the value of the item at the data base setting with the <replacement value>. This usually causes the repositioning of the item within the value set, since value sets are typically ordered in terms of item values. If the data base setting is a cursor, then it is made to "follow" the replaced value, representing a constant positioning with respect to logical associations.

5.3.3.8 Subsetting operators

Subsetting in PRIMDAS can be done in two ways: in one, value sets are used as storage structures into which items from other value sets are duplicated with the TRANSFER operator; in the other method, mark sets are used to keep track of selected items without creating separate copies. In the latter method, the operators MARK and DEMARK are used.

In both cases, data base settings are used to indicate the items to be entered into a subset. The structures which act as the subsets are also to be indicated.

The TRANSFER operator has the form

TRANSFER (<date base setting>,<subset value set>,<mode change>) .

The existence of items in a value subset is independent of those items in the source value set. A value subset is therefore free to contain items from different value sets. This distinguishes it from mark sets which can mark items only from one value set. This freedom, however, requires the resolution of the problem encountered when an item is to be transferred into a value subset having different item attributes from the value set it is in.

Options can be provided for such situations. A default option is to suppress the transfer of items between value sets of differing attributes. Alternatively, the argument <mode change> can be set to indicate the transformation to be carried out in the transfer, if it is necessary to do so. This transformation can be a user procedure.

With mark sets, since no transfer of values is involved, the above considerations are not relevant. The form of the MARK operator is

MARK (<data base setting>,<mark set>) ,

where the mark of the item at the data base setting is entered into the mark set. The mark set is required to be defined on the source set in which the item belongs.

As is the case in value sets, data base settings can be defined on items in a mark set. The deletion of a mark can therefore be specified by

DEMARK (<data base setting>).

5.3.3.9 Inter-value set navigation

Cursors in PRIMDAS are constrained to value sets. Associated items in other value sets may be *accessed* by the ASSOC and AT operators. Also however, it is often required to maintain a position at such an associated item, particularly when it is to be used as a reference point for further data base access.

Rather than allowing cursors to range over different value sets, the approach in PRIMDAS is to provide a further cursor setting operator. The operator sets a cursor defined on the target value set to an item at a specified data base setting, which must also be defined on that value set. This operator is SETAT, which has the form

SETAT (<data base setting>,<target cursor>).

This cursor setting operator enables a wide range of inter-value set navigation. However, a further operation is necessary to represent another commonly required data base traversal. In both indirect and common link associations (Subsec. 5.3.3.3) it is possible for source items to be associated to more than one item in a target value set. It is often required to traverse specifically the set of all those items in a target value set that are associated to particular source items.

The provision of this traversal involves the use of a temporary mark set. An operator, GETLNK, is provided to mark all associated items into a mark set. A cursor can then be defined on this mark set, which

provides in effect, the required traversal. After the navigations have been performed, the mark set can be destroyed.

The form of the GETLNK operator is:

GETLNK (<indirect or common link association setting>,<mark set>).

The items marked are all those determined by the specified indirect or common link association setting. For instance, from Fig. 5.8, the command

GETLNK (<<<L1,SA>,<L2,SB>>,D>,mark set)

would mark the items {d4,d5}. Repeated applications of GETLNK on the same target set, but with different settings, causes the union of the associated items to be marked into the mark set.

The two traversal operations described in this section have retained the simplicity of PRIMDAS cursors while providing for complex navigation of the data base.

5.3.3.10 Structural operators

PRIMDAS structures are used to represent conceptual constructs and associations. On evolution or when particular constructs are to be implemented differently, restructuring of the PRIMDAS representation may be required. Restructuring on PRIMDAS typically can be carried out using PRIMDAS operations themselves. For example, changing the representation of a set of items from a mark set to a value set can be done by creating the required value set into which the mark set items are TRANSFER-ed. Although such operations can be more efficiently implemented within PRIMDAS, their absence would not greatly effect the applicability of PRIMDAS.

There are three basic forms of restructuring of PRIMDAS objects:

(1) mark set ↔ value set

(2) direct association ↔ indirect association

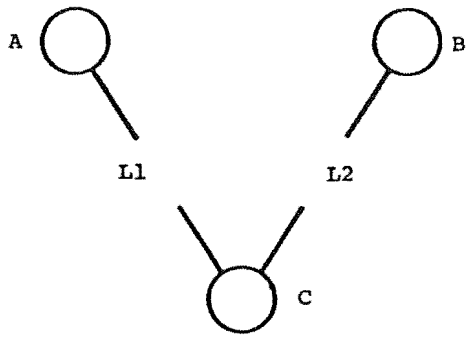
(3) splitting/merging of link structures.

In the change of a value set to a mark set, the source v/m set of the mark set has to be given and checks have to be made to ensure the consistency of the items.

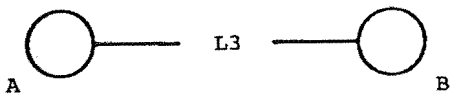
In the second form of restructuring, direct associations are to become represented as indirect associations, through some link. This operation, as well as its inverse, can also be mechanized by a procedure on top of PRIMDAS.

The third form of restructuring given above can also be implemented externally. However, for illustration purposes, they are implemented as PRIMDAS operators in a current implementation (Chapter 7). These operators are described below.

The diagram below illustrates the value sets A, B and C being linked by L1, L2.



It may happen that C is no longer needed, but that the association of A to B embedded within the configuration is to be retained. That is, the following configuration is required:



where L3 is derived from L1, L2. The operator that performs such a derivation is MERGE and the above can be invoked by

MERGE (<L1,L2>,L3)

C, L1 and L2 can then be destroyed.

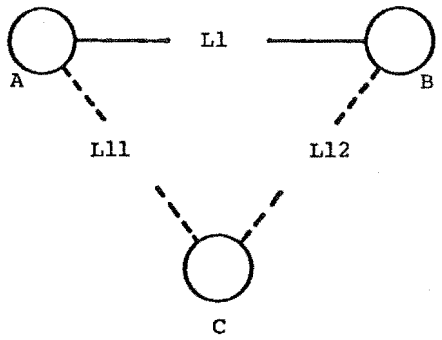
The general form merges a sequence of n links into 1 link:

$$\text{MERGE } (<\ell_1,\ell_2,\ldots,\ell_n>,<\text{target link}>) \ .$$

The links must be such that ℓ_i has a value set in common with ℓ_{i-1} as well as ℓ_{i+1}.

The reverse of MERGE is SPLIT, which will factor a source link structure into two target links. These target links must link each of the value sets on which the source link is defined, to a common value set. Items can then be entered into this common value set using APPEND with a common link setting as described in Sec. 5.3.3.6.

An example is SPLIT (L1,<L11,L12>) which, performed on the following configuration, would factor L1 into L11 and L12. If desired, L1 can then be destroyed.



Either of L11, L12 can be SPLIT further, so that a repeated application of SPLIT will result in a sequence of links.

5.3.4 PRIMDAS Consistency Considerations

Corresponding to the features of implementation base consistency outlined in Sec. 5.2.4, two forms of structural consistency are embedded in PRIMDAS. One is the consistency of subsets represented in terms of mark sets, and the other is the consistency of alternative associational structures represented in terms of link structures. They are described in the following two sections.

5.3.4.1 Consistency of mark sets

Consider the mark set hierarchy of Fig. 5.9. M1 and M2 are mark sets of the value set V; M11, M12 and M13 are mark sets of M1, while M131 and M132 are mark sets of M13. This corresponds to the hierarchy of subsets that is to be maintained.

In specific applications, restrictions may be imposed on where and how values are to enter and leave the hierarchy. However, in the general case, as adopted in PRIMDAS, the entry and exit of data can be specified at any of the mark sets. Each such action triggers appropriate entry or exits in the other sets. These are described below.

Value set deletions are propagated to subsets, so that deletion of an item in V would cause the deletion of any of its marks in any of the mark sets. Two levels of exit can be specified on mark sets. One is a local deletion using DEMARK, which removes a mark from a specified mark set. However, to preserve the subset hierarchy, corresponding marks in any mark set of this target mark set must also be deleted. For example, a DEMARK of a value in M13 would cause that value to also be DEMARKed from M131 and M132. There are no repercussions to the other mark sets. This is in contrast to global deletions, using DELETE, which can also be specified on mark sets. The effect of mark set DELETEs is identical to deletions from the value set V, although they can only be specified on values actually marked in the mark set being operated on.

The entry of data requires the inverse of exit actions, where any input data has to be propagated to all supersets. Local entries are effected by the operator MARK, while global entries are provided for by ENTER. For example, the ENTER of an item into V of Fig. 5.9 causes no further actions. On the other hand, ENTERing into M131 causes the value to be entered into V and its marks put into each of M1, M13 and M131. With MARK, only the marks of existing items can be entered

into the mark sets. It can, however, be defined on any of the supersets of the mark set to be marked. For example, a MARK can be entered into M132 of an item in any of M13, M1 or V. MARKing into M132 from V, however, would cause corresponding marks to be entered into M1 and M13 if they do not already exist.

The above describes the general case of maintaining a subset hierarchy. Specific features can be suppressed in particular situations. An example is where MARKs are only to be specified from immediate supersets. In this, a MARK into M11 can only be of an item in M1 and not of V. Such restrictions are seen as being external to PRIMDAS.

5.3.4.2 Consistency of links

In using the operators MERGE and SPLIT (Sec. 5.3.3.10), the initial and derived links are independent of each other. If the initial links are not destroyed, it is possible for the structures to develop separately. However, if the two sets of links are to act as alternative but equivalent associational structures, then they have to develop together.

A separate set of operators, UNION and PARTITION, is provided by PRIMDAS to specify the equivalence of link structures. UNION and PARTITION are identical to MERGE and SPLIT respectively, except that with the former operators, the initial and derived links are dependent. The form of the consistency to be maintained by PRIMDAS is illustrated below.

Consider the instances in Fig. 5.10, where link L1 between A and B is equivalent to L11 and L12 between A and B. Any associations available between A and B through L1 have to be available through L11 and L12. This means that the delinking of the items pointed to by the cursors SA and SB from L1 would result in the delinkings indicated in the figure. This set of delinkings would also have resulted if a delinking had been specified with respect to SA, SC and L11 or with

respect to SB, SC and L12. This is so since either of these delinkings would cause the loss of the particular association through L11 and L12 between A and B.

Corresponding operations are required in the entry of new associations. An entry into L1 would initiate appropriate entries into L11 and L12, and vice versa. Note that individual entries into either of L11 or L12 are not permitted since neither necessarily specifies an association between A and B.

The consistency considerations discussed above extend to more complex configurations where a series of UNIONs and PARTITIONs may have been carried out.

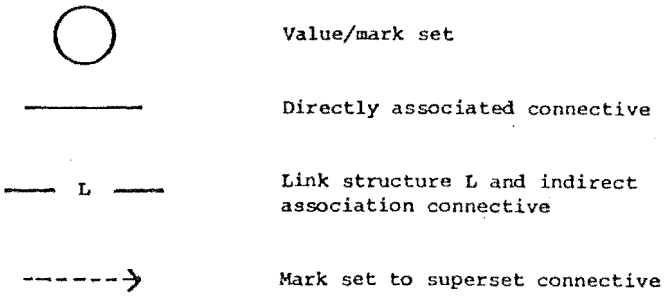


FIGURE 5.1 PRIMDAS diagrammatic representations

APPEND (<value sets of items to be duplicated>,<input value sets>,<source cursor>,<list of cursors>,<source>)

ASSOC (<sequence of link structures>,<source cursor>,<target v/m set>)

AT (<<$link_1$, $cursor_1$>,< >..>,<target v/m set>)

CREATE (LINK, <link structure name>,<v/m set 1>,<v/m set 2>)

CREATE (MARKSET,<mark set name>,<source set>)

CREATE (VALUESET,<list of descriptors>)

DELETE (<data base setting>)

DELINK (<link structure>,<<cursor 1>,<cursor 2>>)

DEMARK (<data base setting>)

DIRASS (<value set list>)

ENDS (<cursor>)

ENTER (<list of value sets>,<list of cursors>,<source>)

FIRST (<cursor>)

GET (<data base setting>)

GETLNK (<indirect or common link association setting>,<mark set>)

LAST (<cursor>)

LINK (<link structure>,<<cursor 1>,<cursor 2>>)

MARK (<data base setting>,<mark set>)

MATCH (<cursor>,<value>)

MERGE (<sequence of link structures>,<target link>)

NEXT (<cursor>)

PARTITION (<source link>,<<link 1>,<link 2>>)

PRIOR (<cursor>)

REPLACE (<data base setting>,<replacement value>)

SEARCH (<value set>,<cursor name>)

SETAT (<data base setting>,<target cursor>)

SPLIT (<source link>,<<link 1>,<link 2>>)

TRANSFER (<data base setting>,<subset value set>,<mode change>)

UNION (<sequence of link structures>,<target link>)

FIGURE 5.2 PRIMDAS operators

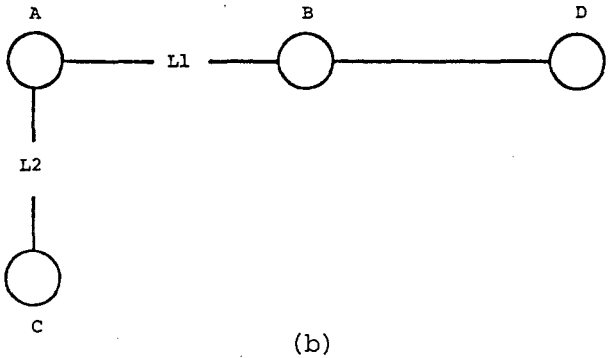
A	
9	a1
2	a2
8	a3
3	a4

B	
6	b1
9	b2
2	b3
5	b4

C	
7	c1
8	c2
1	c3
9	c4

FIGURE 5.3 Direct association

(a)



(b)

A	
6	a1
9	a2
8	a3
2	a4

L1	
A	B
6	23
6	16
8	5
9	5

B	
16	b1
23	b2
5	b3
7	b4

D	
5	d1
26	d2
8	d3
16	d4

FIGURE 5.4 Indirect association
(a) Link configuration
(b) Link entries

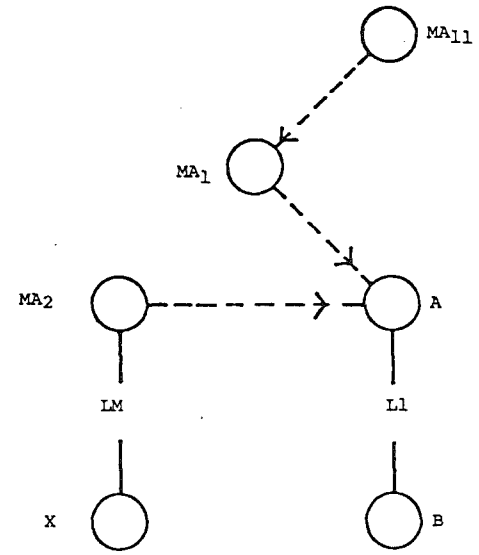


FIGURE 5.5 Mark set configuration

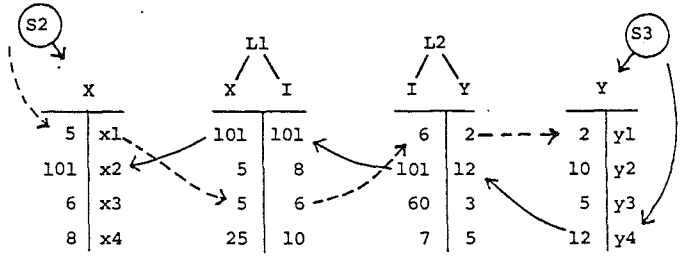


FIGURE 5.6 Indirect associational access

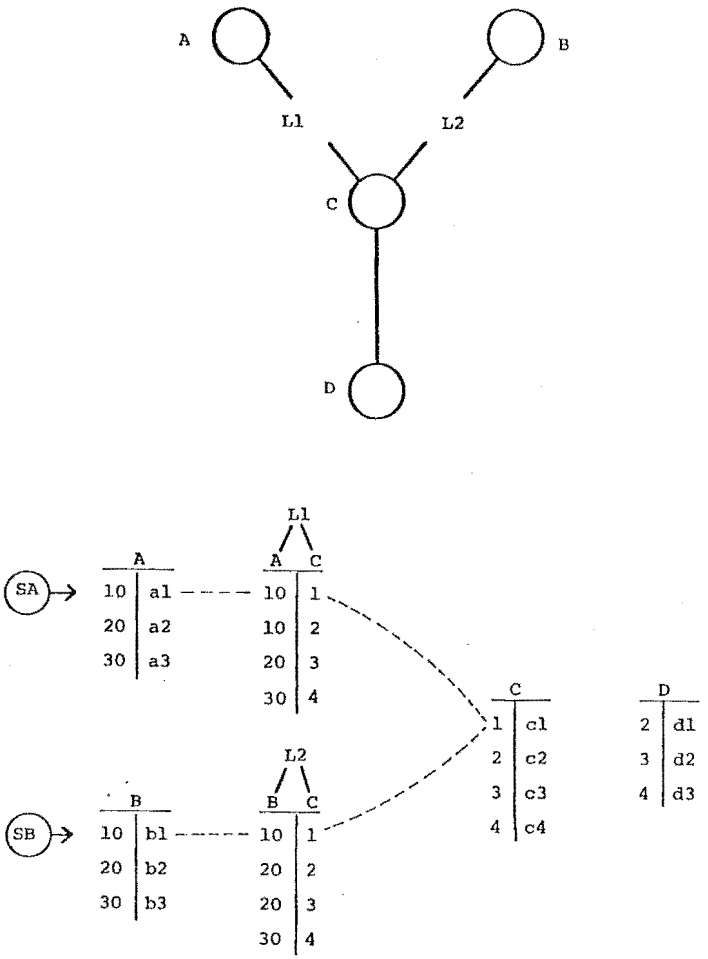


FIGURE 5.7 Common link association

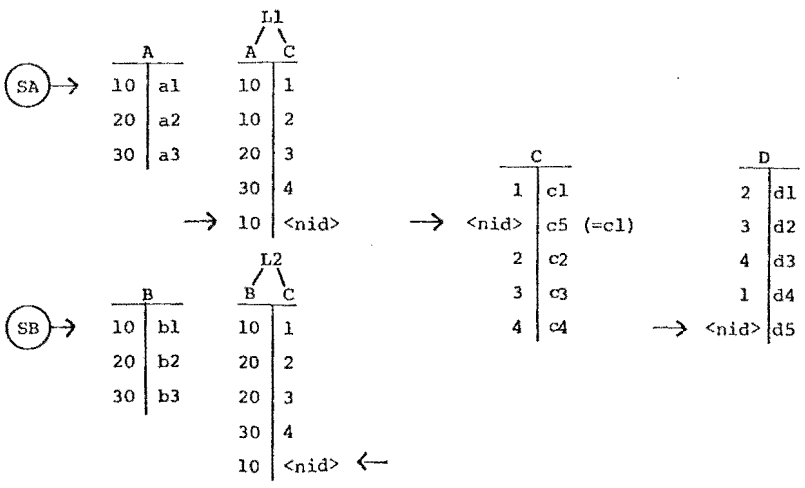


FIGURE 5.8 Specification of common link association

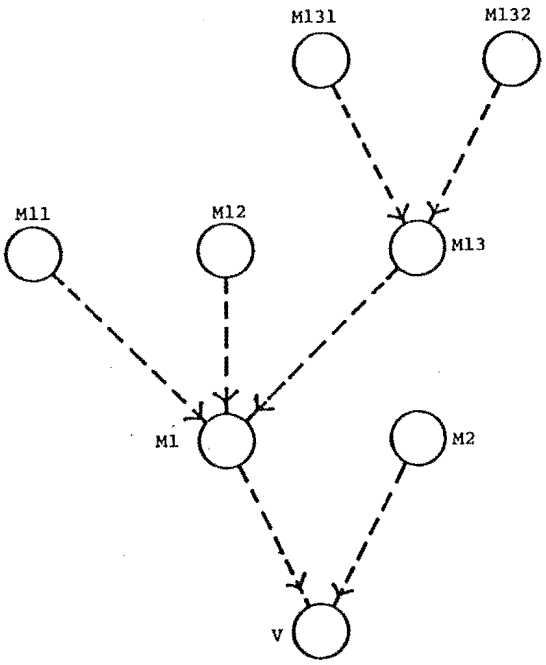


FIGURE 5.9 Mark set hierarchy

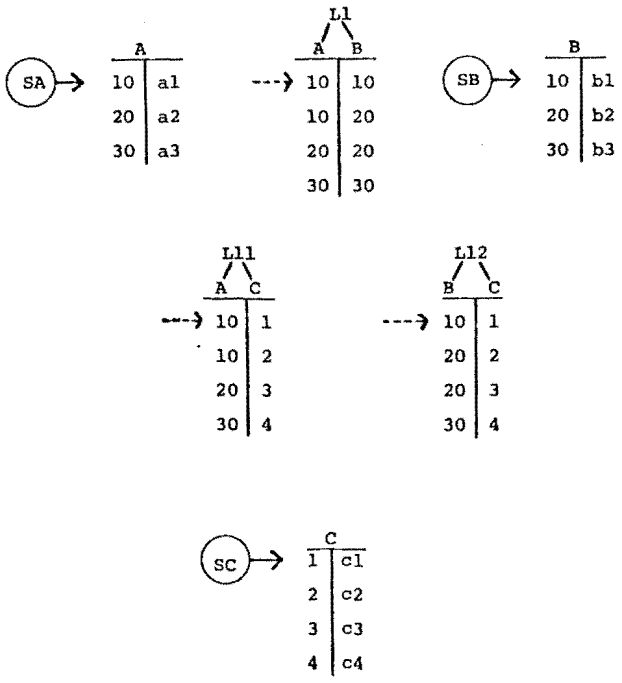


FIGURE 5.10 Delinking of dependent links

CHAPTER 6

REPRESENTATION OF DATA BASE MODELS WITH PRIMDAS

6.1 INTRODUCTION

This chapter illustrates the representation of data base models with PRIMDAS. The roles of PRIMDAS in model support and in model development are discussed in Sec. 6.2.

A specific example of the use of PRIMDAS to implement higher level data representations is given in Sec. 6.3. This section describes PRIMDAS representations of DATAM (Chapter 4) data objects and approaches in the representation of subviews and evolution.

Finally, Sec. 6.4 describes the role of PRIMDAS in supporting coexistence. The approach is illustrated by considering the construction of alternative views of a DATAM data base implemented on PRIMDAS. Alternative views based on the Network and Relational models are outlined.

6.2 MODEL DEVELOPMENT TOOL

6.2.1 Primitive Data Base System

Traditionally, the construction of a data base model involves the construction of a complete data base system solely to support the model. The effort in such model development can be significantly reduced by decreasing the distance from the machine to the model. This can be effected by providing a primitive data base system on which desired models can be constructed.

The relative ease with which further data base facilities can be

constructed on such a system would enable changes or additions to be more readily implemented. This allows an evolutionary approach where applications can maintain user interfaces which adapt to changing needs. That is, changes can be incorporated as the need arises.

This concept of a primitive data base system can be extended to obtain further advantages. On such a system, it would be feasible for a single enterprise to maintain several interfaces. The choice of the data base model to adopt becomes less traumatic since a repertoire may be kept, not only of different models but also of various interfaces to the same model. If a primitive data base system becomes widely adopted, portability of data base models would be greatly enhanced, because the problem of interfacing to a particular machine would have been largely solved by the primitive system. The production of data base system packages can then be approached in two parts: one, the provision of a selection of primitive systems to cater for varying machine and implementation considerations; the other, the provision of data base models for different information situations.

As a generalized data structure interface, PRIMDAS is a suitable basis for such a primitive data base system. The rest of this section discusses the factors involved and the facilities required to augment PRIMDAS in this role as the abstract machine from which data base interfaces are to be constructed. Sec. 6.2.2 provides the perspective with which this role is to be viewed.

A particular facility required in data base systems is that to handle schemata. In multi-level data representations (Chapter 2), there are correspondingly many levels of schemata. Sec. 6.2.3 discusses multi-level schema handling and the role in which PRIMDAS can provide for its support.

6.2.2 The Role of PRIMDAS

The construction of a model and its interface involves building software levels on the implementation base to provide facilities not catered for by the primitive system. These facilities include those considerations of consistency and structure specific to a model, and code to support a particular interface. With PRIMDAS, such construction is aided by some suitable host language.

This host language would typically be some general programming language providing control structures, arithmetic expressions, and so on. PRIMDAS and the host facility together constitute the implementation base, where PRIMDAS serves as the subsystem with which to implement data representation and manipulation. Other than general programming constructs, the implementation base would also require specific PRIMDAS related aids to ease programming effort. This includes facilities to manipulate data external to the data base and control structures based on data base status. A particular example of such aids to augment an Algol-embedded PRIMDAS is given in Appendix B.

The specification of the interface to a data base model includes the form of the data sublanguage. In an evolutionary approach the initial data sublanguage chosen would be a minimal one for the model and application.

Sublanguage interfaces can be seen to consist of two parts (Fig. 6.1): a linguistic level which provides syntactical constructs with which to express data base interaction, and a level of procedures corresponding to the underlying operations required to perform the expressions.

Examples of model construction on PRIMDAS given in this chapter and Chapter 7 do not consider the syntactical forms in which sublanguages are to be provided. Instead, interfaces to models are presented as sets of PRIMDAS procedures implementing the required functions. It is to be

understood that further software may be required to provide suitable user interfaces on these procedures. Different interfaces can be built on a single set of procedures, representing the different ways in which the same functions may be expressed.

6.2.3 Multi-level Schema Handling

In a multi-level model construction, a different schema for the enterprise exists at each level (Chapter 2). The form of the schema which is available for access and manipulation is called the *stored schema*. Each level needs to maintain a stored schema representing the current description of the objects defined on that level.

During the process of structural translation at a level, access of lower level schemata may be required. Therefore, each level needs to provide facilities to interrogate its stored schema. This access is read only, since the responsibility of maintaining it rests only with the particular level.

Each level needs to provide some structure with which to maintain the stored schema for that level. Various approaches can be taken. The structures available at the immediately lower level can be used or structures specifically for the purpose can be constructed. Although the latter approach may be more efficient, it requires more effort to construct.

A common schema structure may be chosen for all the levels. With this approach, only a single schema handling mechanism is required, and there is a standard structural view and access of the schema at each level (Fig. 6.2). The disadvantage is that at each level not having the same structures as that of the schema, two separate structural interfaces are required: one for the data objects supported at that level and another to facilitate schema handling.

Alternatively, on each level the schema can be made to appear as

if constructed with the same objects as that provided at that level.

It is seen that multi-level schema handling is a separate specialized data base problem. The provision of schema handling facilities would further reduce interface construction. PRIMDAS structures and operators can be used for this purpose either directly or as the base on which to construct the facilities.

That is, models built on PRIMDAS can use PRIMDAS data handling facilities to provide schema handling facilities. The implication is that any other function requiring data handling facilities (e.g. subviewing) can also utilize those provided by PRIMDAS.

6.3 REPRESENTATION OF DATA BASE MODELS

6.3.1 Introduction

This section describes the representation of data base models on PRIMDAS. To illustrate the approach, a representation of DATAM is presented.

Functionally, the representation of data base models can be separated into three parts. The first, involving the representation of the objects of the model, and of their associations, is given in Sec. 6.3.2. The second part concerns the representation of the operators of the model and is discussed in Sec. 6.3.3. Finally, the use of PRIMDAS in representing subviewing and evolution is presented in Sec. 6.3.4.

6.3.2 Representation of Data Objects and Associations

The value sets of PRIMDAS provide the data pools, structures in which data is perceived to reside. It is used, therefore, to store the instances of object types. That is, value sets represent instance sets - those collections of items which participate in the data associations that an enterprise is perceived to have at an instant

of time. The associations themselves are represented by any of the three forms of structural associations of PRIMDAS: direct, indirect and common link. Further, mark sets are available for situations in which subsets are required.

To illustrate the application of these concepts, the rest of this section describes the representation of DATAM data objects and of their associations.

Consider the DATAM diagram in Fig. 6.3(a) and a perception of their instances and associations in terms of DATAM tables (Chapter 4) in Fig. 6.3(b).

6.3.2.1 Entity-attribute associations

The EA-table represents the instances and associations of entities to their attributes. These associations can be represented in various ways. However, one approach in particular has distinct advantages over the other. Figs 6.4, 6.5 and 6.6 give alternative representations of the EA-table in Fig. 6.3(b).

In Fig. 6.6, separate value sets for the entity are created for each entity-attribute association. This approach can be immediately rejected on the grounds of the effort required to maintain consistency as well as to perform access.

Alternatively, indirect associations can be used as shown in Fig. 6.5. An advantage here is that instances do not have to be duplicated. However, this approach does not reflect the dependency and "closeness" of attributes to the entity and to each other. In particular, access between C-COLOUR and C-MAKE is through two links.

In the representation of Fig. 6.4, this access, as any other among the entity and its attributes, is direct, requiring no intermediate structures. This representation of EA associations by directly associated value sets is the most appropriate and is adopted here.

6.3.2.2 Relationships

The instances of relationships are represented by the association of the tokens of the two entity-types involved. In the example of Fig. 6.3, the R-table for the relationship between Student and Car is given. For its PRIMDAS representation, it is noted that value sets containing the tokens of Student and Car would already exist in representing its entity-attribute associations. To avoid consistency problems, the relationship can be represented as an indirect association among these value sets, rather than by the creation of further value sets.

The PRIMDAS configuration and instances to represent the relationship are given in Fig. 6.7. The addition of a further relationship between Student and Car just involves the creation of another link structure between the two value sets.

6.3.2.3 Events

In the representation of an event, the association of each event instance to the entities from which it is derived must be available. Also, the relationship between the entities involved must be retained. Both these considerations are catered for by the common link association.

The PRIMDAS representation of the Enrolment event in Fig. 6.3 is given in Fig. 6.8. Event-attribute associations are similar to entity-attribute associations and can likewise be represented by direct associations.

6.3.2.4 Other DATAM concepts

For the DATAM diagram in Fig. 6.3, the complete PRIMDAS configuration is given in Fig. 6.9. The representation of some of the other DATAM concepts (for example dependent entities) are not distinguishable at the PRIMDAS level from the representation of other concepts, in this case from regular entities. Instead some external (to PRIMDAS) means must be provided to ensure that the PRIMDAS objects

are used consistently.

A DATAM concept which utilizes the mark set concept of PRIMDAS is that of a sub-entity. The unique instances of a sub-entity type must at all times be a subset of the unique instances of the source entity type. The existence of the sub-entity instances are determined by their existence as source entity instances. These actions are directly available when using mark sets, where the value set marked represents the instance set of the source entity, and the mark set the instance set of the sub-entity.

6.3.3 Representation of Data Manipulation Operations

The form in which the operation of a model is supported on PRIMDAS is determined by the representation chosen for the data objects and associations of the model. Operators to access and manipulate data in PRIMDAS structures are described in Chapter 5. These operators together with appropriate host facilities provide for the implementation of data base model operators.

To illustrate a particular approach, the rest of this section discusses DATAM operations with respect to the representation described in Sec. 6.3.2.

By DATAM operations, it is meant those access and manipulation operations expressed in terms of DATAM objects and associations. To aid in the discussion, the DATAM operations considered here are loosely grouped into three classes. One class, containing operations involving access of related items, is described in Subsec. 6.3.3.1. Operations in the entry and exit of DATAM objects are considered in Subsec. 6.3.3.2, while 6.3.3.3 presents selected operations requiring more complex PRIMDAS procedures.

A syntax for expressing DATAM operations is not rigorously defined here. Although query languages for DATAM (e.g. at the level

of SEQUEL (Chamberlin *et al.* [1976]), FORAL (Senko [1976a]) etc.) can be constructed, this is not germane to the discussion. Instead, English-like forms are used to indicate DATAM operations, while the corresponding PRIMDAS code is of the form described in Chapter 5.

Also, not all possible operations are given. The selection is designed to illustrate the approach taken in PRIMDAS, in the representation of high level operations.

6.3.3.1 Access of related items

The following operations are considered in this subsection:

1. entity to attribute access
2. attribute to entity access
3. access through a relationship
4. implied access through a relationship
5. access through many relationships
6. access through involvement in an event
7. access of event from member objects

The DATAM diagram and tables of Fig. 6.3 and the corresponding PRIMDAS configuration given in Figs 6.4 - 6.9 will be used to illustrate these operations.

These operations have a common form, where in each, a target instance is to be accessed through some association with respect to specified source instances.

6.3.3.1.1 Entity to attribute access

In this example, given a token of a particular entity, its associated value in an attribute-type of that entity is to be obtained. That is, the access is to obtain the attribute of a given entity.

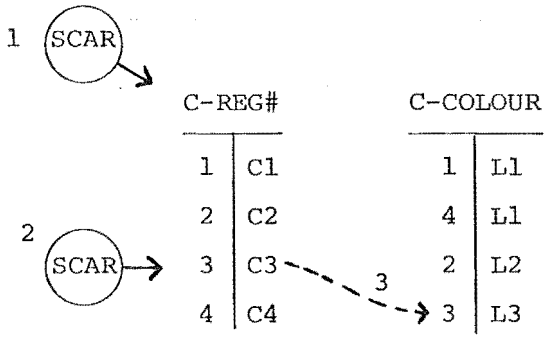
A particular access based on the DATAM diagram of Fig. 6.3 is:

"get the Colour of Car C3"

The PRIMDAS procedure for this involves establishing a position at the item in the C-REG# value set which represents the given Car, followed

by an access from that position to the value set C-COLOUR which contains the associated Colour instances. The access is direct, corresponding to the way entity-attribute associations are represented. The PRIMDAS procedure and actions on the value sets are given below. The numbers indicate corresponding actions.

1. SEARCH (C-REG#,SCAR)
2. MATCH (SCAR,"C3")
3. GET (ASSOC<<>,SCAR,C-COLOUR>)



The access obtains the C-COLOUR item "L3" representing the colour of the car with reg# C3.

6.3.3.1.2 Attribute to entity access

The inverse of the above access is where an entity with a given attribute value is required. The uniform treatment of value sets in a direct association means that the PRIMDAS code to perform the operation

"get the first Car with Colour L1"

is similar to the previous example. This is given below.

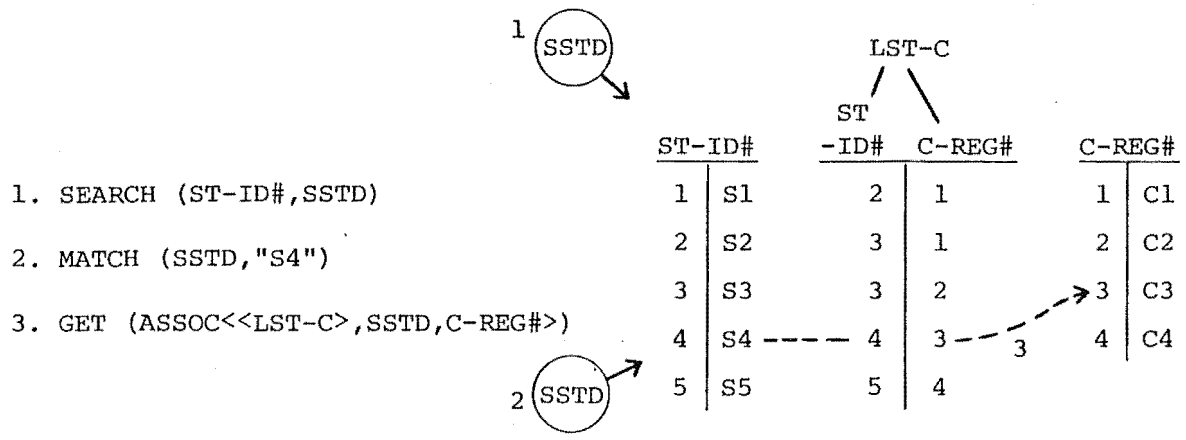
```
SEARCH (C-COLOUR, *)
MATCH  (*, "L1")
GET    (ASSOC<<>,*,C-REG#>)
```

The "*" indicates unnamed cursors.

6.3.3.1.3 Access through a relationship

This example considers the access of an entity associated to a given entity through a relationship. Here access is through the link structure representing the indirect association between the items. For example, below is the procedure to answer the query

"What is the Car owned by Student S4?"



The value obtained, as seen in the diagram is C3 representing the required Car. The reverse access is similar.

6.3.3.1.4 Implied access through a relationship

Implied access is the access of objects not immediately associated, but are related through a sequence of associations. An example of an implied access is that contained in the operation

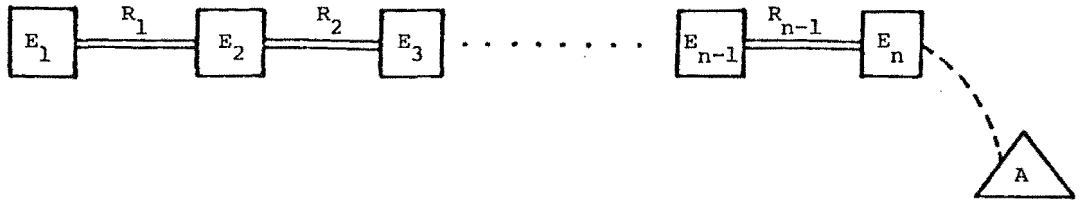
"get the Colour of the Car owned-by Student S4"

where the attribute of an entity related to a given one is required. In the PRIMDAS procedure (given below) the intermediate access of the Car is not needed. This is because entity-attribute associations are represented by direct associations, so that each of the value sets C-REG#, C-COLOUR, C-MAKE are equally accessible. That is, the access is identical to that in the previous example.

```
SEARCH (ST-ID#,*)
MATCH (*,"S4")
GET (ASSOC<<LST-C>,*,C-COLOUR>)
```

6.3.3.1.5 Access through many relationships

A more general form of implied access is one involving a sequence of relationships. Consider the DATAM diagram below.



Then the operation

"get A of E_n R_{n-1} E_{n-1} ..... R_2 E_2 R_1 E_1 = x"

is expressed by including the sequence of links corresponding to the relationships in the access:

SEARCH (E_1,*)
MATCH (*,"x")
GET (ASSOC<<ℓ_1,ℓ_2,...,ℓ_{n-1}>,*,A>)

where ℓ_i is the link structure representing the relationship R_i.

6.3.3.1.6 Access through involvement in an event

Access to items related by involvement in an event involves access through two links. Taking the example in Figs 6.3 and 6.9, the operation

"get a Student taking Course K2"

requires the following PRIMDAS code

```
SEARCH (COURSE-CODE,*)
MATCH (*,"K2")
GET (ASSOC<<LKEV,LSEV>,*,ST-ID#>)
```

Sequences of links are also used when representing access through combinations of events and relationships. For instance, from the model in Fig. 6.3, the operation

"get the Colour of a Car owned by a Student taking Course K2"

is performed by

```
SEARCH (COURSE-CODE,*)
MATCH (*,"K2")
GET (ASSOC<<LKEV,LSEV,LST-C>,*,C-COLOUR>)
```

6.3.3.1.7 Access of event from member objects

The final operation considered in this subsection is that in the access of events given the entities which are involved in them. An example of such an access is

"get the Grade of Student S1 in his Enrolment in Course K2"

This access is directly available with the AT data base setting (Sec. 5.3.3.3).

For the operation above, the code is (illustrated in Fig. 6.8):

```
1.  SEARCH (ST-ID#,SST)
2.  MATCH (SST,"S1")
3.  SEARCH (COURSE-CODE,SK)
4.  MATCH (SK,"K2")
5.  GET (AT<<LSEV,SST>,<LKEV,SK>>,GRADE)
```

The examples in this subsection have shown how access based on associations of DATAM can be represented on PRIMDAS. These accesses can be used, not only to retrieve the items (as indicated in the examples) but also to perform other operations, such as replacement or deletion.

In the construction of a DATAM model interface, PRIMDAS procedures can be made for particular access types that will accept as input the parameters appropriate to that access. A higher level DATAM query facility can then be mapped directly to this set of procedures.

6.3.3.2 Entry and exit of DATAM objects

The entry and exit of the four DATAM objects: attributes, entities, relationships and events are described individually in this subsection. Since the operations on the object-types parallel those on the instances, for brevity the discussion is confined to instances. The DATAM example of Fig. 6.3 is again used for illustration.

6.3.3.2.1 Entry and exit of attribute

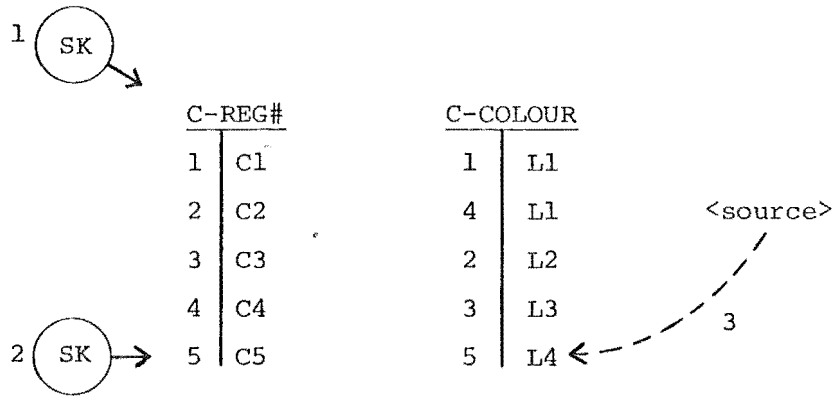
The entry of an attribute into the data base requires the specification of the object which it describes. For instance, the addition of a Colour of the entity, Car, must specify the Car which the Colour describes. In PRIMDAS, this specification is used to determine the Car token instance to which the attribute instance is to be attached. Consider the operation:

"Enter the Colour L4 for the Car C5"

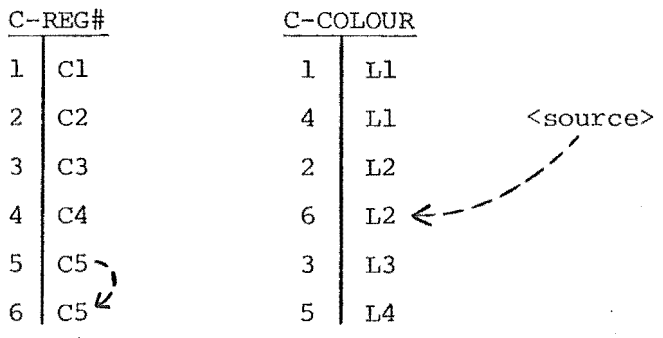
The PRIMDAS code to achieve this is

```
1.  SEARCH (C-REG#,SK)
2.  MATCH (SK,"C5")
3.  APPEND (<C-REG#>,<C-COLOUR>,SK,<>,<source>)
```

where <source> indicates the location and mode of entry of the string "L4". The action corresponding to this is given below.



The parameter <C-REG#> of APPEND indicates that if a Colour already exists for the particular Car, then its token instance is to be duplicated, and to which the new Colour instance is to be appended. In the example, if another Colour, say L2, is to describe Car C5, the following would be entered:



The exit of an attribute has no repercussions on other DATAM objects. It can be represented by any of the forms of access described in Subsec. 6.3.3.1. For instance, the operation

"delete the first Colour of the Car C5"

is mechanized by the code

```
SEARCH (C-REG ,*)
MATCH (*,"C5")
DELETE (ASSOC<<>,*,C-COLOUR>)
```

6.3.3.2.2 Entry and exit of entity

The entry of an entity instance can be done independently of

other DATAM objects. It merely involves the entry of a new token into the appropriate value set. As an example, the operation

"enter the new Car C6"

uses the code

ENTER (<C-REG#>,<>,<source>)

The uniqueness of the token instance has to be established prior to this PRIMDAS operation.

The exit of an entity, on ther other hand, requires consequential actions on other DATAM objects. All attribute instances of that entity have to be deleted, as well as any relationships and events in which it is involved. The operation

"delete the Car C1"

for example, requires the code below to delete the Car's attributes

```
SEARCH (C-REG#,*)
MATCH  (*,"C1")
DELETE (ASSOC<<>,*,C-COLOUR>)
DELETE (ASSOC<<>,*,C-MAKE>)
```

This would be followed by the deletion of the involvement of that Car in the relationship "Student-owns-Car", and completed by the deletion of the token instance C1 itself. The deletion of relationships and events is described below.

6.3.3.2.3 Entry and exit of relationship

The deletion of relationships involves the use of the DELINK operator, since they are represented as entries in link structures. This means that positions have to be established at the two items to be de-associated. In other words, the two objects, entities or events involved in the relationship have to be specified. For example, the following operation

"delete the relationship that Car C1 is owned by Student S2"

can be invoked by

```
SEARCH (C-REG#,SC)
MATCH  (SC,"C1")
SEARCH (ST-ID#,SST)
MATCH  (SST,"S2")
DELINK (LST-C,<SST,SC>)
```

Also, however, in the deletion of an item, any associations it may have are automatically severed. This means that in the deletion of an entity (Subsec. 6.3.3.2.2) its relationships are deleted by the deletion of the token.

The entry of relationships requires both involved objects to be specified. This means that these objects are required to already exist in the data base. The PRIMDAS procedure for the entry of relationships is similar to that in their deletion, with the only difference being that the operator LINK is used instead of DELINK.

6.3.3.2.4 Entry and exit of event

Unlike relationships, events can be deleted without referring to the objects involved, since identification can be by the token of the event. However, frequently the tokens of the objects involved are used to form an event token. In this case, the specification of the event is always in terms of the objects involved. The implication on PRIMDAS is that access of the event can always be via the objects involved. Therefore the tokens of the objects involved do not have to be directly represented on PRIMDAS as tokens of the event instances. Consider the example in Fig. 6.3. The specifications of an event of Enrolment can always be done in terms of the Student and Course involved.

As with entities, the deletion of an event requires the deletion of any attributes it may have, as well as any relationships or events it is involved in. In the example, the operation

"delete the event that Student S1 takes Course K2"

is actioned by the code:

```
        SEARCH (ST-ID#,SST)
        MATCH  (SST,"S1")
        SEARCH (COURSE-CODE,SK)
        MATCH  (SK,"K2")
1.      DELETE (AT<<LSEV,SST>,<LKEV,SK>>,GRADE)
2.      DELETE (AT<<LSEV,SST>,<LKEV,SK>>,ENROLMENT)
```

Operations (1) and (2) represent the deletion of the attribute of the event and of the event itself, respectively. The associations of involvement in LSEV and LKEV are automatically severed.

In the entry of an event, the objects involved must be specified, since events cannot exist in isolation. The PRIMDAS code (given below), for the operation

"enter the event that Student S1 takes Course K3"

involves the use of the AT data base setting with the APPEND operator to enter the token and to link it to the member objects.

```
SEARCH (ST-ID#,SST)
MATCH  (SST,"S1")
SEARCH (COURSE-CODE,SK)
MATCH  (SK,"K3")
APPEND (<ENROLMENT>,<ENROLMENT>, AT<<LKEV,SK>,<LSEV,SST>>,<><source>)
```

6.3.3.3 More complex operations

In the previous subsections, operations have been considered which could be directly represented by PRIMDAS code. In this subsection, more complex operations are considered, such as those operations requiring temporary structures, host facilities or some testing of data base status.

Examples given in this subsection are:

1. Retrieval of instances with values in a given range.
2. Retrieval of all instances of an attribute type for a specified entity.
3. Deletion of all the relationships of an entity in a particular relationship-type.
4. Retrieval of multiple associations through event.
5. Entry of new tokens.

6. Retrieval of a composite attribute.

7. Access of a dependent entity.

In the algorithms, any host facility will be given in terms of an Algol-like pseudocode. These procedures can be adapted into parameter driven routines for each class of operation.

6.3.3.3.1 Retrieval of instances with values in a given range

Any access which returns more than one value requires some form of temporary structure to hold the results. Value sets and mark sets can be used for this purpose, which has the additional advantage of retaining any data base associations. An example is the operation

"get all Students with tokens between x and y"

A mark set, MSUB, is used as well as some comparison facility of the host language. Since the items in value sets are ordered, the procedure involves traversing the value set ST-ID# until a value 'greater or equal' to x is reached, and then marking all unique items until an item greater than y is reached. This is reflected in the following code:

```
SEARCH (ST-ID#,*)
MATCH  (*,"x")
GET    (*)

while item ≤ "y"
      MARK (*,MSUB)
      temp := item
      repeat until item ≠ temp
           NEXT (*)
           GET  (*)
      end-repeat
end-while
```

Note that a convention is adopted here that MATCH (*,"x") puts the cursor at the first item ≥ x. The code does not include considerations when an end-of-value set condition is encountered. The result of the operation can be made available by traversing MSUB.

6.3.3.3.2 Retrieval of all instances of an attribute-type

A similar operation involving the access of attributes is

"get all the Colours of Car C5"

In this case, the marking is of associated items and not of the value set traversed:

```
SEARCH (C-REG#,*)
MATCH  (*,"C5")
GET    (*)
while  item = C5
       MARK (ASSOC<<>,*,C-COLOUR>,MSUB)
       NEXT (*)
       GET  (*)
end-while
```

The while-loop is required since each entity-attribute association is represented with a separate token instance.

6.3.3.3.3 Deletion of all relationships of an entity

In access through link structures, the set of items associated to a given one is obtained by using the GETLNK operator. Consider the operation involving relationships:

"delete all the relationships of Car C1 in Car-owned-by-Student".

The set of Students obtained by GETLNK are then used to establish positions from which the delinking can be specified.

```
SEARCH (C-REG#,SC)
MATCH  (SC,"C1")
GETLNK (<<LST-C>,ST-ID#,SC>,MSUB)
SEARCH (MSUB,SSUB)
FIRST  (SSUB)
while not eov(SSUB)
      DELINK (LST-C,<SSUB,SC>)
      NEXT (SSUB)
end
```

eov(SSUB) is a Boolean which is true when the cursor SSUB is at the end of value set.

6.3.3.3.4 Retrieval of multiple associations through event

Access through an event is similar, except that the GETLNK is defined through two or more links. Also, nested GETLNKs can be used to perform implied access of many items. For example, the operation

"get all the Cars owned by all the Students taking Course K1"

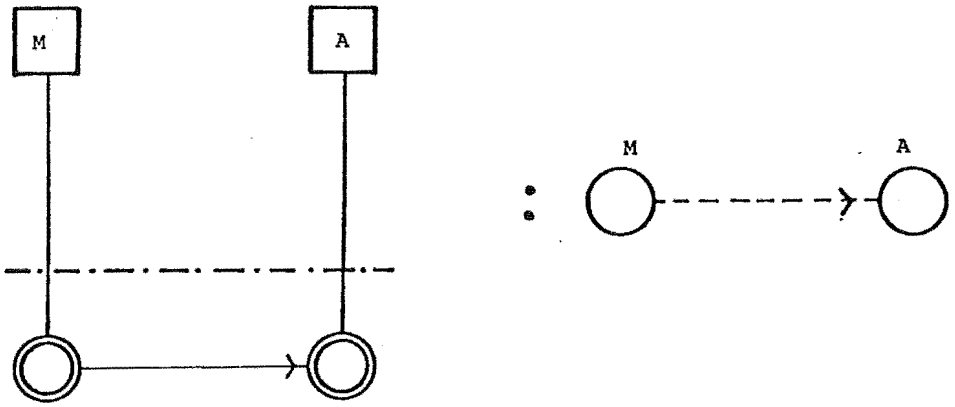
requires two temporary structures, say the mark sets MSUB1 and MSUB2. In the code following, first all the Students taking Course K1 are

found. Then the Cars owned by each Student are marked in MSUB2. Duplicates may occur, but this can be dealt with separately.

```
SEARCH (COURSE-CODE,*)
MATCH  (*,"K1")
GETLNK (<<LKEV,LSEV>,*,ST-ID#>,MSUB1)
SEARCH (MSUB1,*)
FIRST  (*)
while not eov(*)
      GETLNK (<<LST-C>,*,C-REG#>,MSUB2)
      NEXT (*)
end-while
```

6.3.3.3.5 Entry of new tokens

Operations involving the entry of entities into the data base typically requires establishing the uniqueness of tokens. Consider an example (illustrated below) in which the mark set M represents the token instance set of the sub-entity M of the entity A, which in turn is represented by the value set A.



Then the entry of a new token, say x, into A would require the test

```
SEARCH (A,*)
MATCH  (*,"x")
if found then
     non-unique token
else
     ENTER (<A>,<>,"x")
end-if
```

The entry of a new sub-entity, say y, into M, requires additional conditions. If the token to be entered does not already exist in A, then it is ENTERed into M, which will cause the value to be propagated

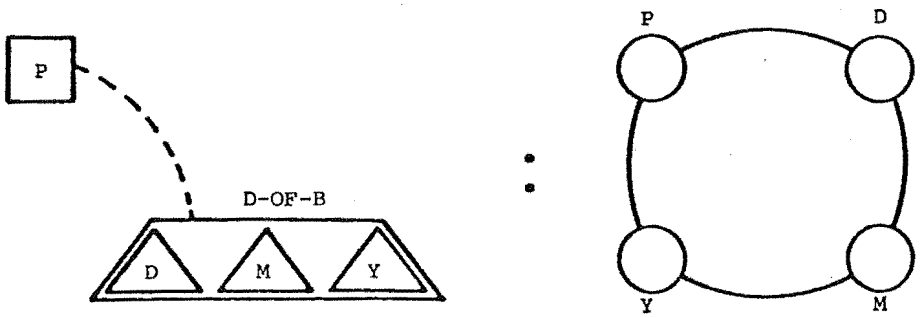
to A. If it is in A, then the mark of y is entered in M and not the value itself. The following is the code representing this:

```
SEARCH (M,*)
MATCH  (*,"y")
if found  then
     non-unique token
else
     SEARCH (A,SA)
     MATCH (SA,"y")
     if not found then
          ENTER (<M>,<>,"y")
     else
          MARK (SA,M)
     end-if
end-if
```

6.3.3.3.6 Retrieval of composite attribute

Other complex operations are those involving objects, the structures of which are maintained by the user. An example of this is the composite attribute. Items in value sets represent the smallest data units that can be manipulated by PRIMDAS. This means that if the components of a composite attribute are to be handled by the system, then each component has to be represented separately in different value sets. However, a consequence of this is that the composite attribute itself has to be effected by the user as procedures on its components.

This is illustrated with an example. Consider the following DATAM diagram and corresponding PRIMDAS configuration:



D-OF-B is a composite attribute of the entity P. Its component attributes are D, M and Y.

The operation on this model of:

"get D-OF-B of P = x"

involves the concatenation of the three associated values of D, M and Y. This is indicated in the following code for the operation:

```
SEARCH (P,*)
MATCH  (*,"x")
GET    (ASSOC<<>,*,D>)
concat item to answer
GET    (ASSOC<<>,*,M>)
concat item to answer
GET    (ASSOC<<>,*,Y>)
concat item to answer
```

(concat is abbreviation for concatenate)

Similarly, any other operation involving D-OF-B requires individual actions on D, M and Y.

6.3.3.3.7 Access of a dependent entity

Another structure which is largely effected by the user is that of the dependent entity. The user has to ensure that the existence of the dependent entity is determined by the existence of the owning entity. Structures have to be provided by the user to remember which particular value sets represent dependent entities and which value sets represent their owning entities. In the access of dependent entities, the user has to ensure that it is done in terms of the owning entity together with some identifying attribute of the dependent entity.

For example, consider the model in Fig. 6.13(a). The operation

"get the B2 of B of A = x that has B1 = y"

contains enough information for a dependent entity access, and can be implemented by

```
SEARCH (A,*)
MATCH  (*,"x")
GETLNK (<<LAB1>,*,B1>,MSUB)
SEARCH (MSUB,*)
MATCH  (*,"y")
GET    (ASSOC<<>,*,B2>)
```

Note that in the PRIMDAS configuration, there is no corresponding value set for B, since B has no tokens. Access can only be done by

identifiers.

6.3.4 Representation of Subviews and Evolution

The provision of object-type subviewing is not directly catered for by PRIMDAS. The user has to maintain structures, perhaps using PRIMDAS objects, to facilitate perception of selected views. Subviewing is typically treated in terms of the objects of the target model. However, if only a PRIMDAS level interface is available to the model, subviewing would have to be specified in terms of the PRIMDAS objects.

With DATAM, the correspondence of the object-types to PRIMDAS objects is virtually one-to-one. The transformation of a subview in terms of DATAM to that in terms of PRIMDAS is a direct mapping. For example, if a subview is not to perceive a particular relationship, then the link structure corresponding to it would be omitted. If a particular entity is not required, then the corresponding value set is made unavailable.

An example of a particular approach to DATAM subviewing is outlined in Chapter 7.

Subviewing of instances requires alternative structures to represent the subsets. These may be mark sets, value sets or link structures, depending on the subviewing. Construction of these subsets is done explicitly using appropriate PRIMDAS operators.

Evolution of data base models may involve restructuring of their PRIMDAS representations. Operators involving specific forms of restructuring are provided in PRIMDAS (Chapter 5). However, where evolution operations are not catered for directly by any PRIMDAS operator, procedures can be constructed to provide the transformations.

To illustrate evolution on PRIMDAS, the following subsections describe approaches in the representation of DATAM evolution operators. The four progressions: attribute → entity, entity → event,

relationship → event and dependent entity → regular entity (Chapter 4) are considered.

6.3.4.1 Attribute → entity

When an attribute becomes perceived as an entity, then the previous entity-attribute association becomes a relationship. On PRIMDAS this means that the direct association between the value sets representing the entity and the attribute becomes replaced by a link structure.

Consider Fig. 6.10. The association through the link structure LAB has to be specified in terms of particular instances of A and B. In the process, any duplications in value set B can be removed by linking those items of A associated to identical values, to a single item. For example, the situation in Fig. 6.11(a) is transformed to that in Fig. 6.11(b). In Fig. 6.11(b), the second occurrences of B1 and B2 are deleted. An algorithm to perform this transformation is given below:

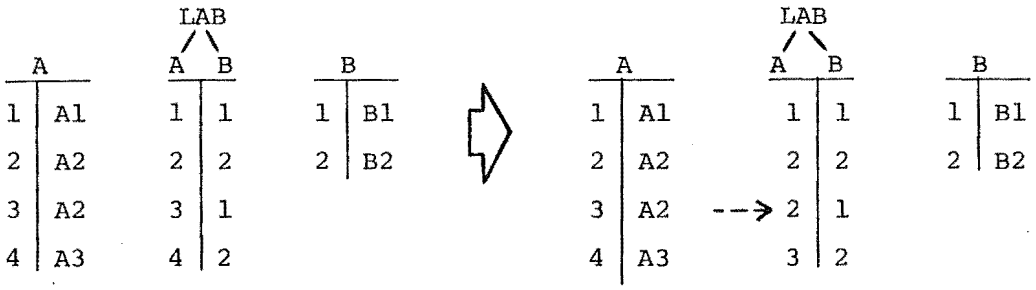
```
          SEARCH (B,SB)
          SEARCH (B,SBKEEP)
          FIRST  (SB)
          FIRST  (SBKEEP)
          SEARCH (A,*)
          while not eov (SBKEEP)
               GET (SBKEEP)
               temp:= item
1.                  SETAT (ASSOC<<>,SB,A>,*)
                    LINK  (LAB,<*,SBKEEP>)
                    NEXT  (SB)
                    GET   (SB)
                    while item = temp
2.                       SETAT (<ASSOC<<>,SB,A>,*)
                         LINK  (LAB,<*,SBKEEP>)
3.                       DELETE (SB)
                         GET    (SB)
                    end-while
               NEXT (SBKEEP)
          end-while
```

The cursor SBKEEP maintains positions at the first occurrences of unique values to be linked to items of A. For this linking, cursors are set at associated items in A by using ASSOC with respect to the SB cursor

(operations (1) and (2)). Note that in deletions (e.g. operation (3)), the cursor becomes updated to the next item in the value set.

An alternative approach is taken in Chapter 7, where an intermediate value set is created and into which the unique values are transferred. The original value set B is then destroyed, followed by the renaming of the intermediate value set to B.

In either approach, if duplicate items occur in A, it is useful if links to items of A are defined on single occurrences of each unique item as well. However, the duplicate items themselves cannot be deleted, since they may be used to facilitate direct associations to some other value set. An illustration of this transformation is given below:



The algorithm for this transformation is similar to that given previously in this subsection. A difference is that here a relinking using DELINK and LINK is performed instead of DELETE.

After this process, the evolution (as approached in this subsection) of an attribute to an entity can be completed by destroying the direct association between the two corresponding value sets.

In the evolution, the IS-A hierarchy in which the attribute is involved is transformed to an IS-A hierarchy among entities. In terms of PRIMDAS, this means that the value sets corresponding to the attributes are changed into a hierarchy of mark sets. Fig. 6.12 gives an example where the attribute A contains B, so that B is evolved to the mark set MB defined on A. The relationship between E and B is then

represented by the link LEB between E and MB. A PRIMDAS procedure, described in Chapter 7 and listed in Appendix C, performs this attribute → entity progression for a given attribute.

6.3.4.2 Entity → event

When a real world event previously modelled as a db-entity is to be evolved to a db-event, then the objects involved in the real world event must also exist in the data base and their involvement to the db-event be specified. The information concerning this involvement has to be supplied by the user since it is not, as yet, contained in the data base.

Consider the DATAM evolution below:

The corresponding PRIMDAS transformation is

In this case, two link structures have to be created to represent the involvement of items of B in the event A. Also, for each unique item a_i of A, the two items of B which are involved in it, say b_j and b_k, have to be provided. For example, assume that this information is input as a sequence of sets of (a_i, b_j, b_k) values. The b_j values represent entities of B involved in a particular *role* (Bachman and Daya [1977]) in the event A, while the b_k values are those involved in the other role of event A. Each role corresponds to separate link structures, so that in the input values, the b_j's represent the association of the a_i's of A to B through, say LAB1, while the b_k's the association through LAB2.

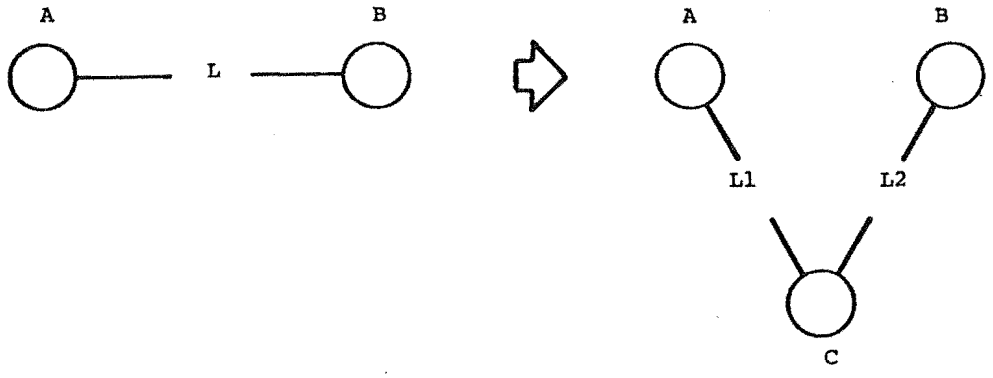
PRIMDAS code to establish the links corresponding to the above

example and input form is given below:

```
SEARCH (A,SA)
SEARCH (B,SB)
while more-input
     input a_i, b_j, b_k
     MATCH (SA, a_i)
     MATCH (SB, b_j)
     LINK  (LAB1,<SB,SA>)
     MATCH (SB, b_k)
     LINK  (LAB2,<SB,SA>)
end-while
```

6.3.4.3 Relationship → event

In the evolution of a relationship into an event, a value set has to be created to represent the token of the event. Corresponding to the PRIMDAS representations of relationships and events, the PRIMDAS transformation of a relationship to an event is illustrated below:



L represents the relationship between the entities represented by the value sets A, B. In the resultant configuration the value set C represents the token instance set of the event. The link structures L1 and L2 are derived from L since the association through L has to be available through L1 and L2.

A PRIMDAS operator is available which represents the above link transformation directly. For the above example, assuming that the link structures L1, L2 and the value set C have been created, the link transformation can be effected by the operation

SPLIT (L,<L1,L2>)

This can then be followed by the destruction of the link structure L

and the input of any tokens or attributes of the event.

Chapter 7 describes and Appendix C lists a PRIMDAS procedure which evolves a given relationship to an event.

6.3.4.4 Dependent entity → regular entity

An example of the evolution of a dependent entity into a regular entity and the corresponding PRIMDAS transformation is illustrated in Fig. 6.13.

The evolution of a dependent entity into a regular entity involves the creation of tokens for the dependent entities. Based on the PRIMDAS representation of dependent entities as depicted in Fig. 6.13, this means that a value set is created to contain the tokens. This value set has to be directly associated to the other value sets representing attributes of the dependent entity. Also the link structure which previously would be defined from the owning entity to one of the identifying attributes has to be replaced by a link structure between the value set of the owning entity and the value set of the new tokens.

Consider the particular example of Fig. 6.13. The entity B, dependent on the entity A through the relationship R_{AB}, is evolved to a regular entity. The value set B represents the tokens for the new regular entity. The relationship R_{AB} now becomes represented by the link structure LAB between value sets A and B rather than the link LAB1 between A and B1. B1 is the value set of the identifying attribute of B.

In terms of instances, this transformation requires tokens in value set B to be appended to the attributes of unique dependent entities. The tokens in B are also to be linked through LAB to the owning entities in A to replace the LAB1 link.

The PRIMDAS transformation described is based on the representations adopted. Alternative representations would require different

transformations. In particular, no restructuring would be required if a dependent entity is implemented identically to a regular entity, with a value set containing dummy tokens. Here, however, the user has to ensure that access with respect to these dummy tokens is suppressed.

6.4 THE ROLE OF PRIMDAS IN COEXISTENCE

6.4.1 PRIMDAS as an Implementation Base

In the construction of an alternative view, two transformations are required: one to create an alternative schema from a subview of the community model, and another to support the interface to the alternative model.

The first transformation is largely a semantic problem. Its main implementation consideration consists of schema handling, in which PRIMDAS can be used to reduce development effort (Sec. 6.2).

The major issue in multiple model support, however, concerns the second transformation, where operations on an alternative view are to be implemented as operations on the actual data base. The transformation involves interpreting the data, which is structured in terms of the community model, as objects of the alternative view. As discussed in Sec. 2.3, this interpretation is best done at a data structure level interface.

The role of PRIMDAS as the basis of an implementation base in the support of independent multiple models is described in Sec. 6.2. The similarity of the considerations there to that in coexistence suggests the applicability of PRIMDAS in the latter as well. The rest of this section illustrates PRIMDAS in the role of implementation base in the support of alternative models. In this role, host facilities, as described in Sec. 6.2, are used to augment PRIMDAS in the formation of transformations.

To illustrate how transformations may be specified on PRIMDAS two examples follow, describing the specification of a network model (Sec. 6.4.2) and a relational model (Sec. 6.4.3) view of a DATAM based community view.

Each of the transformations requires the following information (Sec. 2.3 and Fig. 2.11):

(1) the representation of the community model, DATAM, on PRIMDAS.

(2) the correspondence of DATAM objects to objects in the alternative model.

(3) the form of the operators of the alternative model interface.

The DATAM representation of Sec. 6.3 is assumed in the examples.

6.4.2 A Network View of a DATAM Data Base

In this example, the structural correspondence of DATAM and network objects on PRIMDAS is described first (Sec. 6.4.2.1). The implementation of access on this structure is then illustrated (Sec. 6.4.2.2) by considering various forms of the network FIND operator (CODASYL [1971], Date [1975]).

6.4.2.1 Structural correspondence

Consider the DATAM diagram and tables of Fig. 6.3(a) and (b). With the correspondence to network structures as described in Chapter 4, the data structure diagram for this model is given in Fig. 6.14.

Each of the record-types Course and Student has only single fields corresponding to the token instance sets, while record-types Enrolment and Car have, in addition, fields corresponding to attribute instance sets.

The PRIMDAS configuration of the DATAM diagram in Fig. 6.3 is given in Fig. 6.9. Not all of the network structures of Fig. 6.14 have a direct PRIMDAS representation.

Each of the record-types Course, Student and Car represent entity-attribute associations, and corresponds to the appropriate set of directly associated value sets. The record-type Enrolment represents an event-attribute association, and corresponds to the PRIMDAS value sets ENROLMENT and GRADE, which represents the token and attribute instance sets of the db-event Enrolment. Also, the two set-types Ce and Se represent the involvement of the entities Course and Student in the event Enrolment, therefore corresponding to the PRIMDAS link structures LKEV and LSEV respectively.

However, the record-type Sc-link and the set-types Ls and Lc, representing the relationship Student-owns-Car, together correspond to the single PRIMDAS link structure LST-C. To avoid the complexity of simulating network access on Sc-link, Ls and Lc by access on LST-C, further PRIMDAS structures can be created to correspond more closely to the network structure.

The PARTITION operator is used to represent the link LST-C with an alternative construct consisting of two link structures and a value set. This results in the configuration of Fig. 6.15, where the links LS and LC are equivalent to LST-C. These additional structures correspond directly to the network structures; the links LS, LC to the set-types Ls, Lc and the value set SC-LINK as the only field in the dummy record-type Sc-link. No consistency problems arise, since the equivalence of the link LST-C and the two links LS, LC is maintained by PRIMDAS. The link LST-C is retained to facilitate DATAM access of the relationship.

The representation of relationships is the only case in the correspondence where the addition of structures would be advantageous.

6.4.2.2 Network access

Network access on the configuration will be illustrated by considering the CODASYL DML operator FIND, which constitutes a major

part of the access operations in the DBTG-DML.

The following subsections will establish first the approach taken in representing currency concepts: *current of set, current set occurrence* and *current of run-unit*. This is required in the subsequent descriptions of the FIND operator.

Representations of four forms of the FIND operator will be given:

(1) FIND record-type USING key

(2) FIND NEXT record-type OF set-type

(3) FIND OWNER RECORD OF set-type

(4) FIND OWNER IN set-type 1 OF CURRENT OF set-type 2

6.4.2.2.1 Currency indicators

FIND establishes data base positions by setting *currency indicators*, which on PRIMDAS can be represented by setting PRIMDAS cursors. Access with FIND is determined either by some key (identifier, data base key, CALC-key) or relative to some previously determined data base position. Where access is with respect to a key, MATCH can be used to locate the desired item. It is to be noted that the LOCATION MODE IS CALC clause is not relevant here, since no control is permitted on the storage of items.

The setting of a PRIMDAS cursor to a single value set item is in effect a record setting, since all the other items in the record are available through ASSOC. Therefore, a currency indicator on a record-type can be represented by a cursor on any one of the value sets representing the record-type.

In PRIMDAS, more than one cursor may be defined on any record-type or set-type. This means that the RETAINING CURRENCY clause (Taylor and Frank [1976]) can be achieved by creating and using a temporary cursor for the operation.

In the correspondence of link structures to set-types, a direction has to be imposed on the links to indicate the owner-member

association. Also, the *current of* set indicator, which is set to the most recently referred to record in a set, can be either a member record or an owner record. To represent this currency, two cursors are required for each set. One cursor would be defined on the owner record, while another would be defined on the member record. The currency indicator for the set is the cursor most recently referred to among the two. For set currency purposes, these two cursors are not independent. Another cursor in conjunction with GETLNK is required to represent the actual *current set occurrence* and associate the current of set cursors. These will be illustrated in the examples below.

Therefore to represent the currency indicators of the data base as seen by an application program, a cursor is required for each record-type and for each set-type in the data base view. Any particular currency indicator is then represented by one of the cursors. For example, the *current of run-unit* is that record at which the most recently referred to record-type cursor is positioned. The currency indicator itself is actually the cursor, but since it completely determines the current record or set occurrence, in the following discussion cursors will be used to refer to both currency indicators and occurrences interchangeably.

Consider the PRIMDAS configuration in Fig. 6.15 and its network view given in Fig. 6.14. During initiation of a data base session, cursors would be set on each of the value sets COURSE-CODE, ST-ID#, C-REG#, ENROLMENT and SC-LINK, representing the currency indicators on the corresponding record-types. Additionally, a mark set is required for the member value set of each link that represents a set-type, so that traversal through the set-types can be mechanized. Cursors on these mark sets would then represent the current set occurrence indicators.

Let the mark sets MLKE, MLSE, MLS and MLC be defined on the value sets ENROLMENT, ENROLMENT, SC-LINK AND SC-LINK respectively.

Also, let the cursors on the data base be SCRS, SS, SCAR, SE, SL, SMLKE, SMLSE, SMLS and SMLC on the value/mark sets COURSE-CODE, ST-ID#, C-REG#, ENROLMENT, SC-LINK, MLKE, MLSE, MLS and MLC respectively. This is illustrated in Fig. 6.16.

The configuration in Fig. 6.16 is used in the descriptions of the representations of four forms of FIND described in the following sub-sections.

6.4.2.2.2 FIND record-type USING key

This first FIND format locates a particular record based on some value of its keyfield. An example is:

FIND CAR USING AKEY

where the key is contained in the identifier AKEY. The PRIMDAS code to represent this merely involves setting the cursor SCAR to the item with the value in AKEY:

MATCH (SCAR, AKEY)

After each such operation, the run-unit currency indicator is set to the resultant cursor, in this case SCAR. Also, SCAR becomes the currency indicator of the set Lc, in which Car is the owner.

Consider the instances as in Fig. 6.4 and that AKEY contains the value C1. The invocation of a DBTG GET operator following the FIND statement would retrieve, using ASSOCs, the items <C1,L1,M1>. This represents the current run-unit occurrence pointed to by SCAR.

6.4.2.2.3 FIND NEXT record-type OF set-type

This second FIND format steps through member records of a set, an example of which is

FIND NEXT SC-LINK RECORD OF LC SET

This operation requires that all those records of Sc-link which are associated to a record of the owner, Car, have been determined.

Consider the case that this operation follows the FIND operation in 6.4.2.2.2 which sets the currency indicator of Lc to SCAR. To get to

the next record occurrence in the set, the member records are first retrieved into the mark set MLC by (EMPTY (MLC) removes all previous marks from MLC):

```
EMPTY  (MLC)
GETLNK (ASSOC<<LC>,SCAR,SC-LINK>,MLC)
```

The currency indicator is then set by the commands

```
FIRST (SMLC)
SETAT (SMLC,SL).
```

The first command sets the cursor SMLC at the first of the member records. The second sets the cursor of the actual record-type, SL, to this occurrence. SL, the cursor on SC-LINK, then represents the current of set Lc, the current Sc-link occurrence as well as the current of run-unit.

Any subsequent invocations of this FIND operator requires only:

```
NEXT  (SMLC)
SETAT (SMLC, SL)
```

The end of set occurrence error status would be set when an end-of-value set condition occurs in stepping through MLC.

6.4.2.2.4 FIND OWNER RECORD OF set-type

This FIND operator sets a currency indicator at the owner record of a set. An example is

```
FIND OWNER RECORD OF LC SET
```

Consider the invocation of this operation following the FIND operations of the previous two subsections. The cursor SMLC represents the current of Lc. The implementation of the above FIND operation involves setting the cursor of the Car record-type. This is done by

```
SETAT (ASSOC<<LC>,SMLC,CAR>,SCAR)
```

The current of run-unit, the current of Car and the current of Lc are set to SCAR.

The actual current of Lc is an occurrence of Sc-link. However, SMLC can be used as above, since it determines the current of Lc. The cursor SMLC in conjunction with the mark set MLC in effect also keeps

track of the current Lc occurrence. This is so since MLC contains all record occurrences of the current set occurrence while maintaining a link to the owner in Car.

6.4.2.2.5 FIND OWNER IN set-type 1 OF CURRENT OF set-type 2

The last FIND operation considered here is to locate an owner where the reference is made in terms of different set types common to a member record-type.

An example is

FIND OWNER IN LS OF CURRENT OF LC SET

The link structure corresponding to the set-type Ls is LS and the record to be located is an occurrence of Student. That is, the cursor SS is to be set at an appropriate item of value set ST-ID# through the link LS. The current of Lc, with respect to which the record of Student is to be located, is, as noted in the previous subsection, available through the cursor SMLC.

PRIMDAS code to perform the FIND operation is

SETAT (ASSOC<<LS>,SMLC,ST-ID#>,SS)

The currency indicators which are to be set to SS following this command are the current of Student and the current of run-unit.

6.4.3 Relational View of a DATAM Data Base

6.4.3.1 Structural correspondence

The objects of the relational model consist of table structures containing the instances of data and their associations. A correspondence of DATAM objects to functional/multi-valued dependencies is given in Chapter 4. However, this requires a further composition process to construct the relations themselves. For clarity, this will be avoided and the correspondence will be taken to be the viewing of DATAM tables (Chapter 4) as relations.

Consider the DATAM diagram in Fig. 6.3(a). The relational

objects in the alternative view correspond to the EA, Ev and R-tables in Fig. 6.3(b) which represent the perception of DATAM data instances. Seen as relations, particular columns of each table have to be chosen as the prime key. In EA and Ev tables, the columns of token instances can be used as a key if all the object-attribute associations are *n:1*. If not then columns of those attributes with *n:m* associations also need to be included in the key. If all else fails, all the columns together can always be taken as the key.
In R-tables, if the relationship is *n:1* then one of the columns can be the key, otherwise both columns have to be used.

For the DATAM diagram of Fig. 6.3, the corresponding relations are given in Fig. 6.17, with the instances as in Fig. 6.3(b).

These relations are to be interpreted on the PRIMDAS configuration (Fig. 6.9) of the DATAM diagram. Relational attributes having the same domains in different relations correspond to the same value set. For example, the columns Car in Student-owns-Car and that in Car-relation both correspond to the value set C-REG#. Also, the representation of each relation corresponds to the configuration representing the DATAM concept from which it is derived. For example, the relation Student-owns-Car corresponds to the db-relationship of the same name and is represented by the value sets ST-ID# and C-REG#. On the other hand, the tuples of Car-relation are represented as PRIMDAS direct associations, since it corresponds to entity-attribute associations.

6.4.3.2 Relational access

6.4.3.2.1 Tuple access

Access of tuples are determined by the PRIMDAS configuration of each relation. For example, the retrieval of a Car-relation tuple involves direct-association, corresponding to the representation of entity-attribute associations. For example, the relational retrieval

"get tuple of Car-relation with Car = C1"

can be actioned by the PRIMDAS code

```
SEARCH (C-REG#,*)
MATCH  (*,"C1")
put "C1" into answer
ASSOC (<>,*,C-COLOUR)
concat item to answer
ASSOC (<>,*,C-MAKE)
concat item to answer
```

On the other hand, access of relations derived from R-tables would require indirect association. This is because the associations of the items are represented by PRIMDAS links. For instance, the retrieval:

"get tuple of Student-owns-Car with Student = S2"

can be implemented by the code

```
SEARCH (ST-ID#,*)
MATCH  (*,"S2")
put "S2" into answer
ASSOC  (<LST-C>,*,C-REG#)
concat item into answer
```

In this case, however, the tuple is not completely determined, as the association is *n:m*. Since the key involves all relational attributes, the retrieval of a particular tuple requires knowing the tuple itself. This is reasonable, since such a relation would typically be used more in join operations (Codd [1970]) rather than tuple access.

Access of tuples of relations derived from Ev-tables are correspondingly different and can be implemented with common link associational access. An example is the operation

"get tuple of Enrolment-relation with Course = K1 and Student = S1"

PRIMDAS code for this is

```
SEARCH (COURSE-CODE,SC)
MATCH  (SC, "K1")
SEARCH (ST-ID#,SS)
MATCH  (SS, "S1")
GET    (AT<<LKEV,SC>,<LSEV,SS>>,ENROLMENT)
put item into answer
concat "K1" into answer
concat "S1" into answer
GET    (AT<<LKEV,SC>,<LSEV,SS>>,GRADE)
concat item into answer
```

6.4.3.2.2 Alternative representations

Access can be simplified by creating structures to represent all the relations in a standardized form. This however, means that further considerations are required to maintain consistency in the additional structures.

This trade-off between simplicity of access and minimising consistency considerations is also apparent in the implementation of relational operators e.g. *join, projection* and *restriction.* The construction of separate PRIMDAS structures to represent derived relations could be such that any consequent access or operation becomes simple to perform. On the other hand, the consistency of these structures is typically not maintained by the DATAM implementation, so that it becomes the responsibility of the alternative view implementation.

The converse of this approach is where derived relations are interpreted on the PRIMDAS configuration without creating any redundant data. As an example, consider the representation of the restriction operator. A possible implementation is to utilize a mark set to remember the selected tuples or rows of a restricted relation. The advantage of this is that no considerations of consistency are required. This is because, as a mark set, the consistency of the alternative structure is determined by the value set on which it is defined. However, this approach results in yet another PRIMDAS representation of a relation. That is, a further algorithm for tuple access is required.

The choice is determined by the overall system goals. It should be noted, however, that typical relational interfaces (Astrahan *et al.* [1976], Date and Codd [1974]) contain no direct reference to the traditional relational operators join, projection, and restriction. Furthermore, typically the operations available at these interfaces can be represented directly on PRIMDAS. Therefore there is no need for the traditional operations to be implemented as a mandatory level in a

multi-level relational approach.

6.4.3.2.3 Direct representation example

As an example of a situation in which access usually effected by a relational operator is directly available on PRIMDAS, consider the equijoin operator.

In a join, a relation is constructed from two others based on a common domain. The columns of the resultant relation are the union of the source relations, while the rows are determined by using the common column as a pivot.

In a relational view of DATAM, relations derived from R-tables are, by their nature, often used in join operations. Consider two relations derived from R-tables, given in Fig. 6.18(a). Fig. 6.18(b) gives the PRIMDAS configuration and instances on which the relations are represented. The natural join of these two relations (given in Codd [1970]), is the relation

R * S (	Supplier	Part	Project)
	S1	P1	J1
	S1	P1	J2
	S2	P1	J1
	S2	P1	J2
	S2	P2	J1

Exactly these tuples are available from the PRIMDAS structures through the two links, so that no further structure is required. For instance, the operation

"get a tuple of R * S with Supplier = S2 and Part = P1"

can be mechanized by the PRIMDAS code

```
SEARCH (SUPPLIER,*)
MATCH  (*, "S2")
GETLNK (<<LSP>,*,PART>,MSUB)
SEARCH (MSUB,*)
MATCH  (*, "P1")
ASSOC  (<LPJ>,*,PROJECT)
put    "S2" into answer
concat "P1" into answer
concat item into answer
```

which would return the tuple <S2, P1, J1> as required.

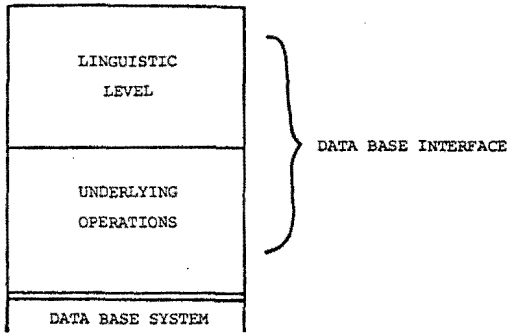


FIGURE 6.1 Software levels in data base sublanguages.

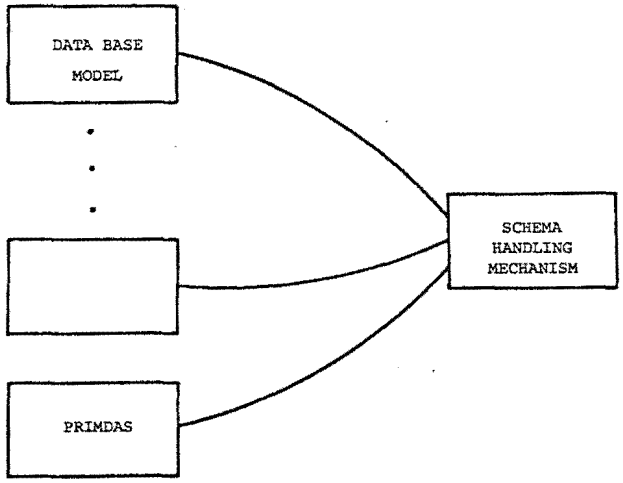
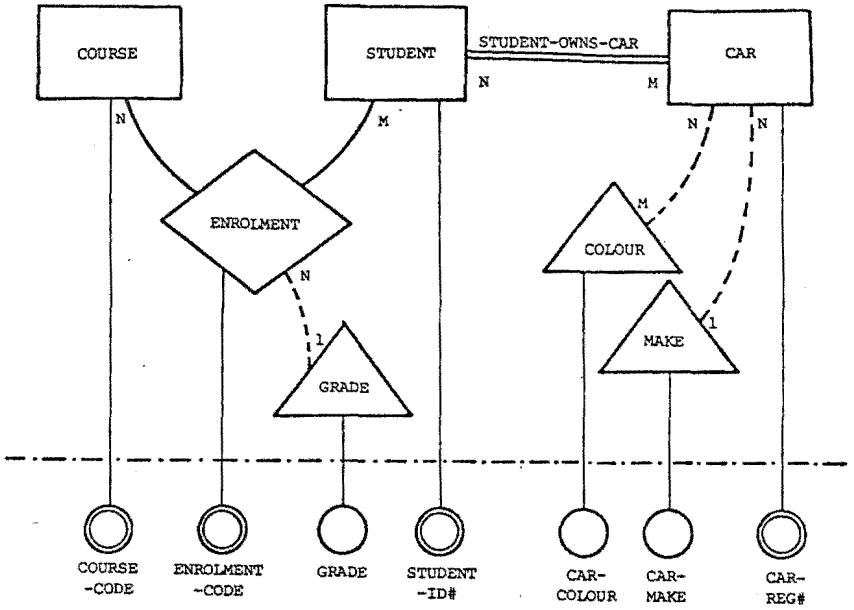


FIGURE 6.2 Multi-level schema handling

(a)



(b)

EA-TABLE

CAR	COLOUR	MAKE
C1	L1	M1
C2	L2	M2
C3	L3	M1
C4	L1	M3
C5		M4

R-TABLE

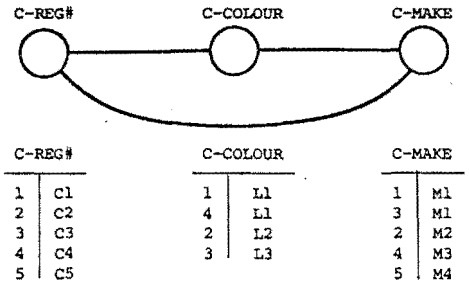
STUDENT	CAR
S2	C1
S3	C1
S3	C2
S4	C3
S5	C4

Ev-TABLE

ENROL-MENT	COURSE	STUDENT	GRADE
V1	K1	S1	G1
V2	K1	S2	G2
V3	K1	S3	G1
V4	K2	S1	G3
V5	K2	S4	G4
V6	K3	S2	G2

FIGURE 6.3 Data base example

(a) DATAM diagram
(b) DATAM tables

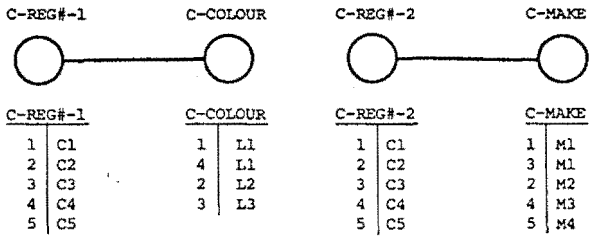


C-REG#	
1	C1
2	C2
3	C3
4	C4
5	C5

C-COLOUR	
1	L1
4	L1
2	L2
3	L3

C-MAKE	
1	M1
3	M1
2	M2
4	M3
5	M4

FIGURE 6.4 Direct association representation of EA-association



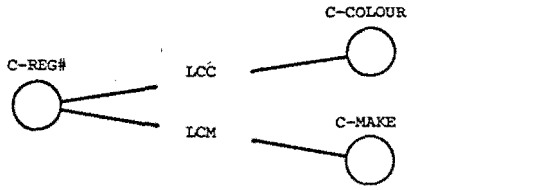
C-REG#-1	
1	C1
2	C2
3	C3
4	C4
5	C5

C-COLOUR	
1	L1
4	L1
2	L2
3	L3

C-REG#-2	
1	C1
2	C2
3	C3
4	C4
5	C5

C-MAKE	
1	M1
3	M1
2	M2
4	M3
5	M4

FIGURE 6.6 Alternative EA-representation



LCC

C-REG#	C-COLOUR
10	1
20	2
30	3
40	1

C-COLOUR	
1	L1
2	L2
3	L3

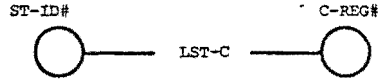
C-REG#	
10	C1
20	C2
30	C3
40	C4
50	C5

LCM

C-REG#	C-MAKE
10	1
20	2
30	1
40	3
50	4

C-MAKE	
1	M1
2	M2
3	M3
4	M4

FIGURE 6.5 Indirect association representation of EA-association



ST-ID#	
1	S1
2	S2
3	S3
4	S4
5	S5

LST-C

ST-ID#	C-REG#
2	1
3	1
3	2
4	3
5	4

C-REG#	
1	C1
2	C2
3	C3
4	C4

FIGURE 6.7 Representation of relationships

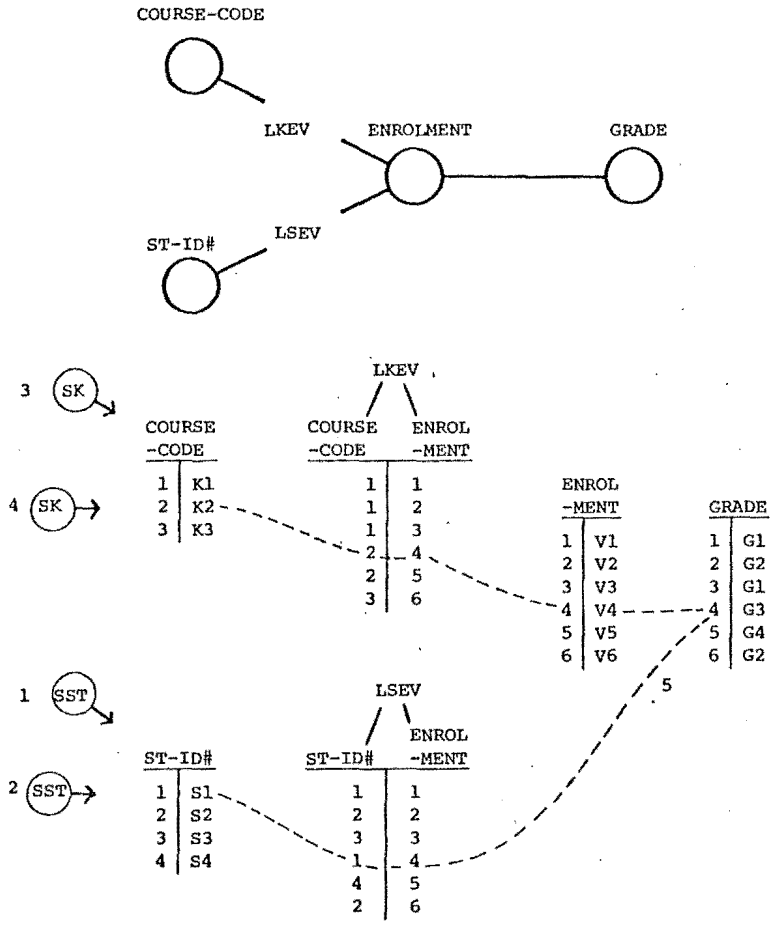


FIGURE 6.8 Event representation

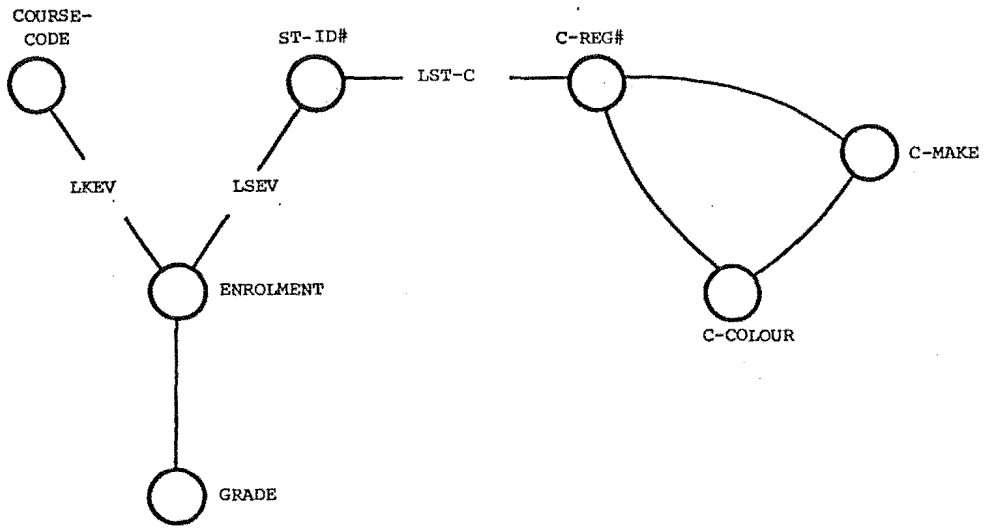


FIGURE 6.9 PRIMDAS configuration for Figure 6.3

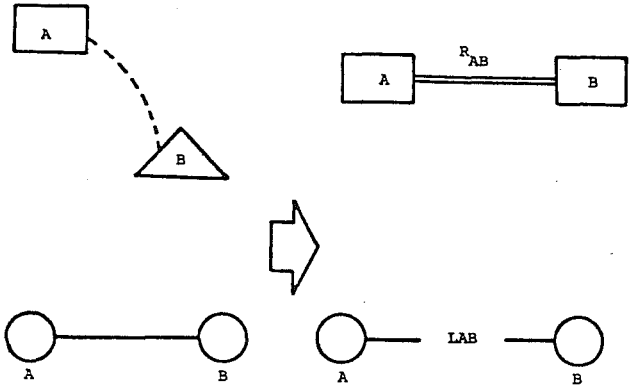


FIGURE 6.10 Attribute association → relationship

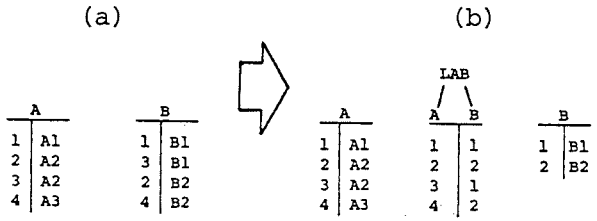


FIGURE 6.11 Direct association → indirect association

(a) direct association
(b) indirect association

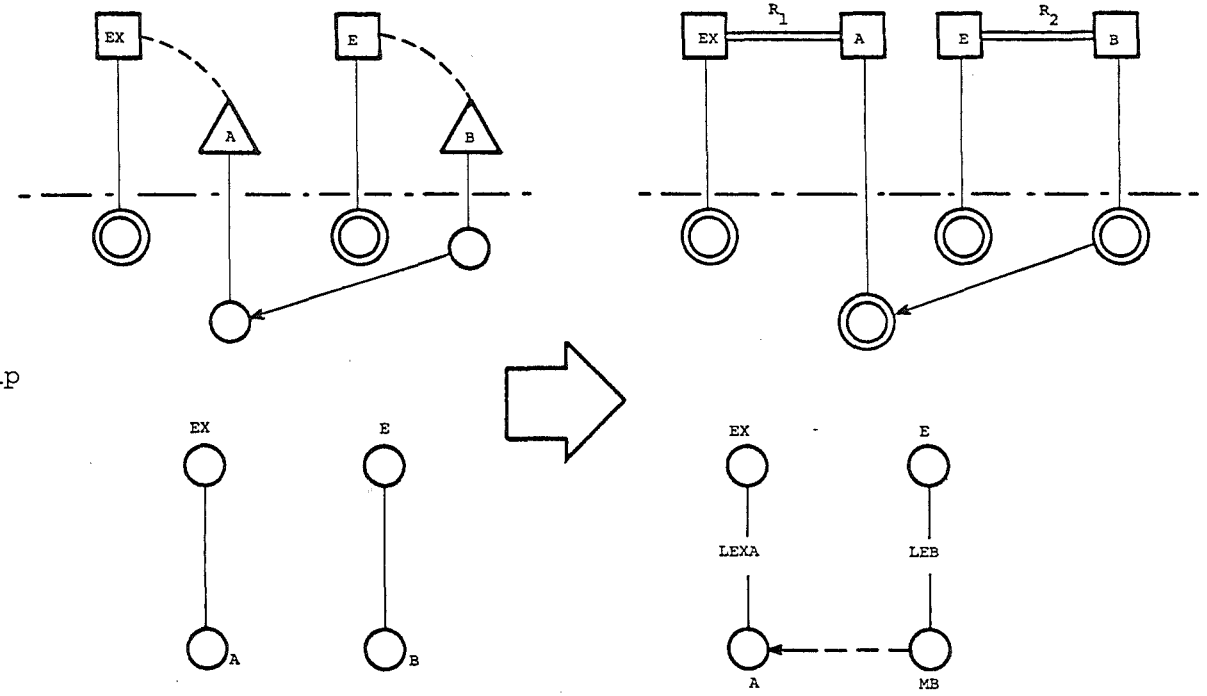


FIGURE 6.12 Representation of evolution of conceptually contained attributes

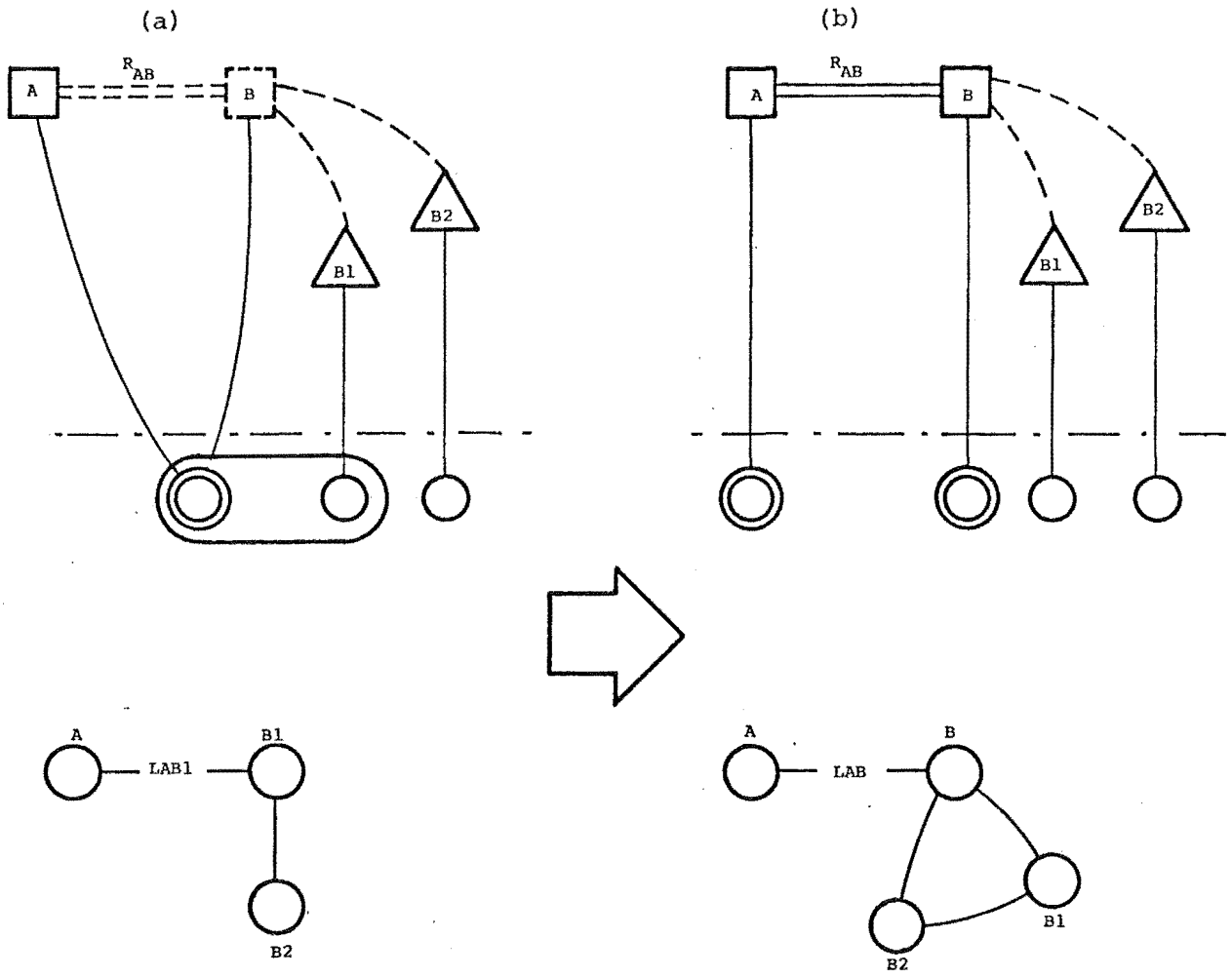


FIGURE 6.13 Dependent entity → regular entity

(a) Dependent entity
(b) Evolved regular entity

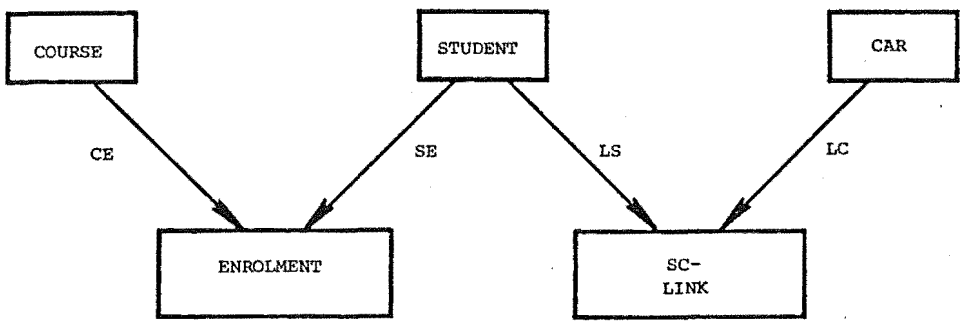


FIGURE 6.14 Network view of Figure 6.3

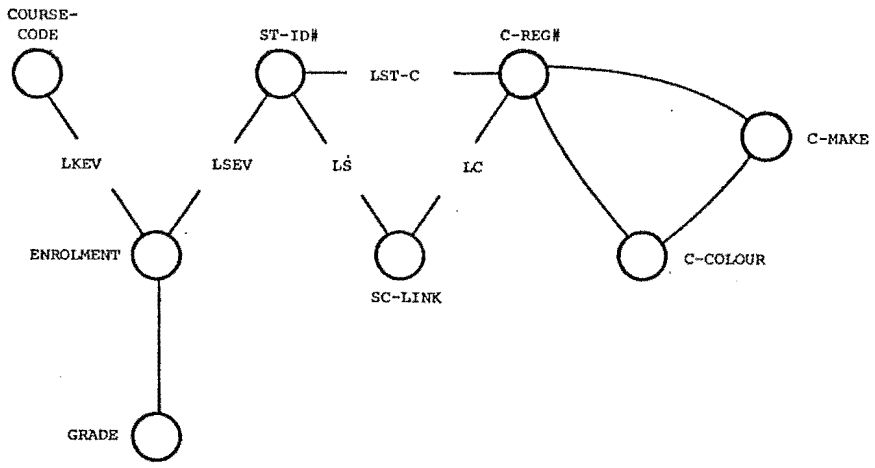


FIGURE 6.15 PRIMDAS configuration to support network access of Figure 6.14

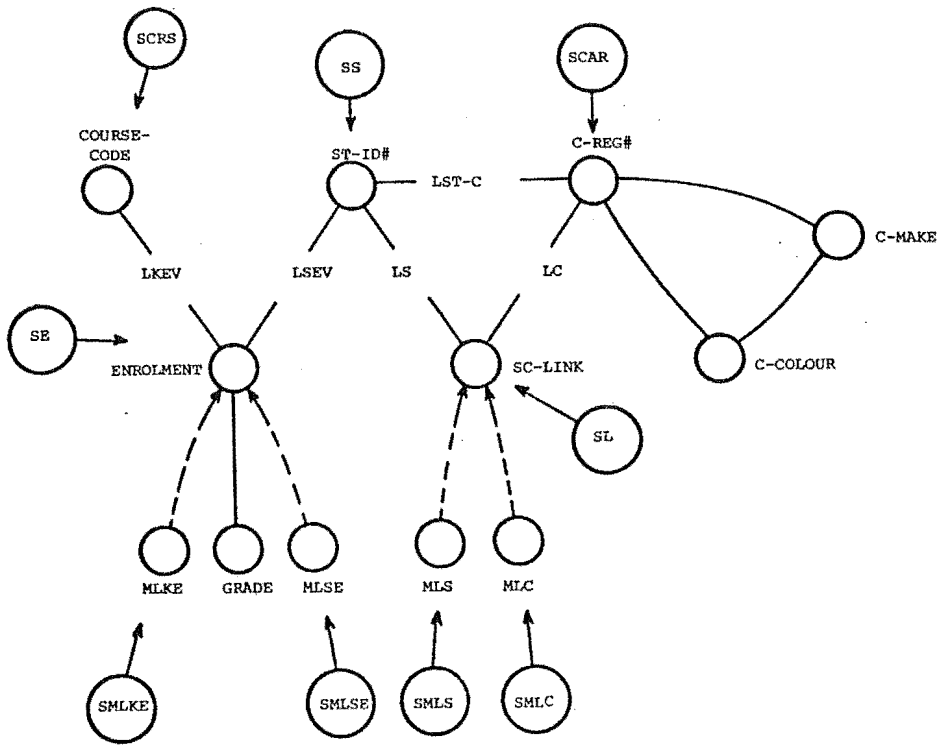


FIGURE 6.16 Configuration to support currency indicators

ENROLLMENT-RELATION (ENROLLMENT,COURSE,STUDENT,GRADE)

STUDENT-OWNS-CAR (STUDENT,CAR)

CAR-RELATION (CAR,COLOUR,MAKE)

FIGURE 6.17 Relational view of Figure 6.3

(a)

R (SUPPLIER	PART)
S1	P1
S2	P1
S2	P2

S (PART	PROJECT)
P1	J1
P1	J2
P2	J1

(b)

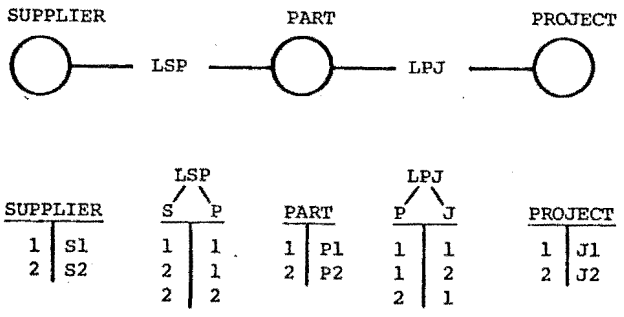


FIGURE 6.18 Illustration of join

(a) Relations

(b) Corresponding PRIMDAS configuration

CHAPTER 7

AN EXAMPLE - DATA BASE MODEL AND DATA BASE CONSTRUCTION

7.1 INTRODUCTION

Chapter 4 describes a specific data base model, DATAM, and Chapter 5 presents a primitive data structure interface. The form in which DATAM objects and operators can be supported on PRIMDAS is described in Chapter 6. This chapter expands on this by considering the support of a DATAM system on a specific PRIMDAS implementation.

Provision for schema handling (Chapter 6) is an integral part of model support. The exposition of this chapter centers around a particular approach in schema handling. Sec. 7.2 describes the schema structures used for PRIMDAS and DATAM. The implementation of DATAM operators requires structural information and is therefore based on these schema structures. The support of the schema structures can be viewed as a data base construction on PRIMDAS, with the DATAM operators as procedures to maintain them. Examples of the implementation of DATAM operators, with emphasis on their interaction with the schema structures are given in Sec. 7.3.

These operators effectively form a DATAM interface. In Sec. 7.4, their use in the construction and manipulation of a small DATAM data base is outlined. Since the schema structures contain all structural information, they also form the bases for evolution operations and alternative view construction. Examples of evolution operations are presented in Sec. 7.5, while an approach in supporting alternative views is given in Sec. 7.6, using a network view for illustration.

The DATAM operators described in this chapter have been

implemented on a PRIMDAS system implemented on a Burroughs B6700. Details on the implementation and code for the procedures described in this chapter are available in the Appendices.

7.2 SCHEMA STRUCTURES ON PRIMDAS

7.2.1 PRIMDAS Schema Structure

The operators of a data base model are constructed as procedures of PRIMDAS operators. PRIMDAS acts as a primitive data base system handling all data access and manipulation of the higher level model. In the construction of the operators of the model, information on the PRIMDAS structures used is required to describe the correspondence of the model objects to them. For this purpose, PRIMDAS maintains a sub-data base which contains information on the PRIMDAS structures used in the data base. This PRIMDAS *schema structure* is itself constructed from PRIMDAS objects and its configuration is given in Fig. 7.1. The names are prefixed by '0' indicating a level 0 schema structure which is available as read-only to PRIMDAS users.

In the configuration, the value set OFILE contains all the objects used, which are directly associated to their descriptors OFTYPE, OFLNGTH, etc. The structure OVMSET, which contains all the value and mark sets used, is a mark set of OFILE and is linked to itself by ODIRAS to record the existing direct associations. OVSET, OMARK and OLINK are ultimately mark sets of OFILE, containing respectively, the value sets, mark sets and links used. The association of value/mark sets by links and the links between particular pairs of these objects are modelled by the substructure containing the value set OL, the mark sets OVMSET, OLINK and the links OLWITH, OLTO and OHASL. The complete PRIMDAS configuration can be viewed as representing the DATAM diagram given in Fig. 7.2.

In this implemention, a CREATE-MODEL command is provided to initialize the PRIMDAS schema structure. This must be invoked prior to any data base construction. The schema structure is maintained by PRIMDAS, where entries and deletions are triggered by various PRIMDAS operators which update the schema.

7.2.2 DATAM Schema Structure

To maintain DATAM objects and to specify their correspondence to PRIMDAS objects, DATAM also needs a structure in which to store its schema. Any DATAM data description facility, or generally any operation on DATAM objects, involves manipulation of this further sub-data base. The DATAM diagram and PRIMDAS configuration of this DATAM schema structure are given in Figs 7.3 and 7.4 respectively.

This structure records the DATAM concepts used and the PRIMDAS objects which represent it. In this implementation, the correspondence between DATAM and PRIMDAS objects is as described in Chapter 6. Also the correspondence is direct in that no change of names is involved - so that, for example, an entity 'A' has its token instance set represented by a value set, 'A'. The name, 'A', would then be a member of the mark set lENTY which contains all value sets or mark sets which represent token instance sets. Details concerning the functions of this configuration will be discussed in Sec. 7.3 in the context of DATAM operators which use it. A complete description is available in Appendix B.

The names of the PRIMDAS objects in the DATAM schema model begin with the character "1" to differentiate them from the PRIMDAS schema objects and from the enterprise objects which constitute the real data base. This provides the mechanism to enable the structure to be made read-only to any DATAM users. The "1" could be seen to indicate that DATAM is a data representation one level above PRIMDAS.

From Fig. 7.3 it is seen that various level 1 objects, e.g. 1DESOB, 1RELN, are mark sets defined on level 0 objects. This does not violate the read-only aspect of the PRIMDAS schema objects and is in fact the mechanism for correspondence.

Construction and manipulation of this sub-data base is effected by the direct use of PRIMDAS as a primitive data base system.

The PRIMDAS command, CREATE (Sec. 5.3.2), is used to construct the mark sets, value sets and links which constitute the structure. The commands used to construct the DATAM schema structure are given in Fig. 7.5. Each command is passed to PRIMDAS as a string containing the pertinent parameters. While the general form of the commands is as described in Chapter 5, the exact format and conventions used in this particular implementation are as given in Appendix A. For example, in Fig. 7.5, the first three commands construct the value sets 1RANGE, 1TYPE and 1LNGTH, while the last command directly associates them.

The passing of these commands to PRIMDAS constitutes the initialization operation in the construction of a new DATAM data base.

7.3 DATAM CONSTRUCTION

Chapter 6 describes the representation of DATAM operations in terms of PRIMDAS operations. It does not however, indicate how the correspondence between DATAM and PRIMDAS objects or any other structural information is made available. To obtain such information requires access of the DATAM schema structure (Fig. 7.3) and/or the PRIMDAS schema structure (Fig. 7.1).

This section gives examples of the use of the schema structures in the storage and retrieval of the structural information of DATAM data bases. The examples include descriptions of procedures representing DATAM operators. These procedures form the base on which more user

oriented interfaces can be built.

The following subsections describe the particular usage of the schema structures with respect to different classes of operators. Procedures to enter range descriptions into the schema structure are described in Subsec. 7.3.1, while 7.3.2 describes examples of procedures to create DATAM object-types. Procedures in the entry of DATAM instances are given in 7.3.3. Subsec. 7.3.4 discusses procedures in the exit of DATAM object-types and instances. Finally, 7.3.5 describes examples of DATAM retrieval routines. The descriptions of specific procedures are brief. References are made to code listings in Appendix C.

7.3.1 Range Procedures

In the declaration of ranges, information is provided specifying attributes of each range. These are then used to ensure that data input to an instance set is consistent to that of the range on which it is defined. A rigorous high level range specification facility is described by McLeod [1976]. In the current PRIMDAS implementation, however, only the basic attributes of type (integer, real or alpha) and character length are considered.

An example range declaration is given below:

```
DECLARE
RANGE  CAR-REG
       TYPE    ALPHA;
       LENGTH  6 CHAR;
END
```

Such information is to be entered into the schema structure of Fig. 7.3. This is done by the CRANGE procedure, the actual code of which is given in Fig. C.3(a). The DATAM schema structure objects relevant to this procedure are the value sets lRANGE, lTYPE and lLNGTH. lRANGE contains the range names, while their type and length are stored in lTYPE and lLNGTH respectively. These latter value sets are both directly associated to

lRANGE.

The action of entry can be simply performed by

ENTER (<lRANGE,lTYPE,lLNGTH>,<>,<source>)

However, the uniqueness of range names has to be ensured. Thus CRANGE also includes code to test that the name input to the routine is not already in use.

In the specification of a DATAM enterprise model (Chapter 4), the IS-A associations among attributes are to be indicated as associations among the ranges on which they are defined. A more complete range declaration would therefore contain this information:

e.g.
```
DECLARE
     RANGE  COLOUR-USED-IN-CAR
          TYPE     ALPHA;
          LENGTH   15 CHAR;
     SUPER-RANGE   COLOUR;
     SUB-RANGE     CAR-BODY-COLOUR,
                   CAR-INTERIOR-COLOUR;
END;
```

which represents the specification of the COLOUR-USED-IN-CAR attribute of Fig. 3.10.

The association of a range to its super-range, such as between Colour-used-in-Car and Colour above, is represented through the link lCONTN. The inverse of lCONTN represents the associations of a range to its sub-ranges. Routines to enter these links, SETSUPR and SETSUBR, are given in Figs C.3(b) and C.3(c).

7.3.2 Create Procedures

To illustrate the procedures used to create DATAM objects, this section describes the entry of db-entities, sub-entities and attributes into the enterprise model.

7.3.2.1 Entity construction

In the schema structure (Fig. 7.3), the mark set lDESOB contains all those value or mark sets which represent the instance sets of describable objects. lENTY, in turn a mark set of lDESOB, represents

all those items of lDESOB which represent instance sets of entities. The construction of a DATAM entity therefore consists of creating a value set to represent its instance set and then marking it into lENTY. In addition, however, the range of the instance set must also be recorded. This association is represented by the link lOBRNG between lDESOB and lRANGE.

PRIMDAS procedures to perform these manipulations are given in Figs C.4(a) and C.4(b). It corresponds to declarations of the form

```
DECLARE
     ENTITY  name
          HAS-RANGE  range-name
END .
```

7.3.2.2 Sub-entity construction

An example of a subentity declaration form is

```
DECLARE
     SUBENTITY  name 1
          OF    name 2

END
```

where name 1 is declared to be a sub-entity of name 2. The DATAM schema objects lSUBOB, lHASUB are used to represent this information. lSUBOB contains all those mark sets which correspond to the instance sets of sub-entities, while the link lHASUB associates these sub-entities to the entities on which they are defined.

The algorithm to construct a sub-entity is:

```
procedure  CONSTRUCT-SUB-ENTITY (SUBA);
begin
     create mark set SUBA representing its instance set;
     mark   SUBA from OVMSET into lDESOB;
     mark   SUBA from lDESOB into lSUBOB;
     link   the mark in lSUBOB to its source entity in
            lDESOB, through lHASUB;
end;
```

CRSNTY, listed in Fig. C.5, is a PRIMDAS procedure to perform this construction.

7.3.2.3 Attribute construction

Procedures to construct attributes are more involved since more

structural information is to be recorded. This includes the object that an attribute describes and the cardinality of mapping of the attribute association. The schema structure representing attribute association is more complex than those of entities and therefore their manipulation procedures more involved.

In Fig. 7.3, the value set lATTRI contains all attribute names. The link lCMPOS from lATTRI to itself links an attribute to those of which it is composed, if any. Since composite attribute names do not correspond to any value set, the set of attribute names is not a sub-entity of the set of PRIMDAS structures and consequently is not represented as a mark set.

The association of an attribute to the object it is describing is recorded in the links lAA and lAOBJ. The value set lHASAT contains unique numbers, each of which represents an attribute association. This attribute association is described by the directly associated value set lANUM, which contains its cardinality of mapping. The link lATRNG associates an attribute to the range on which it is defined.

Declaration of an attribute would contain the description of the attribute and the descriptions of its members, if any. For a non-composite attribute, a possible declaration format is:

```
DECLARE
ATTRIBUTE name 1
          OF describable object name
          CARD-MAP  cardinality of association
          HAS-RANGE range name
END
```

Note that the IS-A associations of the attribute are represented by the associations of conceptual containment of its range (Sec. 7.3.1).

The declaration of a composite attribute would be a sequence of these individual attribute declarations:

```
DECLARE
ATTRIBUTE name 0
     OF describable object name
     CARD-MAP cardinality of association
     MEMBERS
          [attribute 1
          [attribute 2
               :
          [attribute n
END
```

The component attributes would not have OF nor CARD-MAP clauses, since it would subsume those of the composite attribute.

PRIMDAS procedures to perform attribute construction are given in Fig. C.6. There are two main procedures, CRATTR and STCMPO (Fig. C.6(a)). The first enters the basic information of a non-composite attribute, while the latter constructs an attribute as the composite of existing attributes.

The algorithm for CRATTR to construct attribute A of the object D, is outlined below:

```
procedure  CONSTRUCT-ATTRIBUTE (A, D);
begin
     create value set A, representing the attribute instance set;
     directly associate A to the PRIMDAS structure, D, of the described
          object;
     enter the name A into lATTRI and
          link it through lAA and lAOBJ to the mark of D in lDESOB;
               in this same operation, a unique entry representing
               the attribute association is made in lHASAT;
     append the cardinality of mapping in lANUM to this new entry in
          lHASAT;
     link the item A in lATTRI to its range in lRANGE through lATRNG;
end;
```

In the construction of a composite attribute (STCMPO), a value set is not created. Instead the composite attribute name is entered in lATTRI and then linked through lCMPOS to those attributes of which it is composed.

Further DATAM object construction routines are described in Appendix C. These are procedures to create relationships and events.

7.3.3 Enter Procedures

Enter procedures are those to enter instances of existing objects

and associations. These procedures are of various forms, determined by the mode of input. For example, the bulk entry of data would be handled differently to that of the single inputs typical of interactive updates. In both cases, however, it must be ensured that no rules are violated in the entry of any instances. This validation may require access of the DATAM schema structure (Fig. 7.3).

An example in which no schema access is necessary, is in the input of new tokens. Here the only consideration is to ensure the uniqueness of the tokens. This can be determined by using the MATCH operator as indicated in the code in Subsec. 6.3.3.3.5.

Validation in the entry of attribute instances, on the other hand, involves access of the schema objects. In particular, the cardinality of the attribute association has to be retrieved. The status of the data base is then checked to ensure that the entry of the attribute instance does not transgress this cardinality.

The cardinality of attribute associations are stored in the value set lANUM of Fig. 7.3 and can be retrieved jointly from lATTRI and lDESOB through the links lAA and lAOBJ. This access is indicated in the procedure below. The parameters to the procedure are the attribute-type (ATT), the object it is an attribute of (OBJ), the instance to be entered (INS) and the token to which the instance is to be appended (TOK).

```
procedure ENTER-ATTRIBUTE (ATT,OBJ,INS,TOK);
begin
     SEARCH (lDESOB,SD)
     MATCH  (SD,OBJ)
     SEARCH (lATTRI,SA)
     MATCH  (SA,ATT)
     GET (AT<<<lAA,SA>,<lAOBJ,SD>>,lANUM>)
     card-map:= item
     ENDS (SA)
     ENDS (SD)
          SEARCH (OBJ,*)
          MATCH  (*,TOK)
          if card-map not violated then
               APPEND (<OBJ>,<ATT>,*,<>,INS)
          else
               error ("cardinality violated")
          end-if
          ENDS(*)
end;
```

Listings of PRIMDAS code to perform various forms of enters are given in Appendix C. This includes routines to enter relationships and events.

7.3.4 Exit Procedures

Exit procedures consist of destroy and delete operations. Destroy operations remove object types from the data base and involve deleting entries from the schema. DATAM delete operations, on the other hand, remove only instances and do not update the schema. Both types of operations, however, require access of the schema to determine the objects that are to be removed as required by the existence rules of the abstractions.

For example, the exit of a describable object type requires the exit of all of its attribute types, all of its sub-objects, and any relationship or event type in which the object is involved. Since an event is itself a describable object, this last associated destroy means that a procedure to destroy describable objects is nested.

The schema structure (Fig. 7.3) is accessed to retrieve the attributes, relationships and events of an object to be destroyed. An object destroy procedure therefore involves access of a significant portion of the schema. Following is an outline of a procedure to destroy

a describable object (OBJ).

```
procedure  DESTROY-DESCRIBABLE-OBJECT (OBJ);
begin
     get attributes of OBJ by accessing lATTRI from lDESOB through
           the links lAOBJ and lAA;
          comment this access, as well as those below, involves
                creating a temporary mark set and using GETLNK;
          destroy the attributes;
     get the relationships in which OBJ is involved by accessing
           lRELN (contains relationship names) from lDESOB
           through lRTO, lROBJ and lRR;
          destroy the relationships;
     get the sub-objects of OBJ by accessing lSUBOB from lDESOB
           through lHASUB;
          destroy the sub-objects;
     get the events in which OBJ is involved, by accessing lEVENT
           (contains event names) from lDESOB through lEVOBJ and
           lEVEV;
          destroy each of the events by invoking
                DESTROY-DESCRIBABLE-OBJECT(event_i);
     destroy the actual PRIMDAS structure of OBJ;
end;
```

A PRIMDAS procedure to perform this destroy is listed in Appendix C (Fig. C.13).

While the destroy of a describable object may involve the exit of other objects, the destroy of an attribute type involves only the exit of the attribute itself. Consequently, the procedure is significantly simpler. In terms of the schema structure, the exit of an attribute type involves deleting its cardinality of mapping in lANUM, the attribute association represented in lHASAT, and the attribute name entry in lATTRI. The only PRIMDAS structural operation required is that to destroy the value set representing the instance set of the attribute. A procedure to destroy attribute types is given in Fig. C.15.

Deletion procedures are similar to destroy operations, except that associated instances are removed rather than associated object types. Access of the schema structure is required to determine which object instances are to be deleted in any consequential actions. As with destroys, deletions of describable object instances involve wide ranging access of the schema, while attribute deletions do not require

any schema access. Procedures to delete objects and attributes are given in Figs C.16 and C.17, respectively.

The destroy and deletion of other DATAM objects similarly involves access of appropriate structures of the schema. The access is determined by the rules embedded in the abstraction and by the function of particular DATAM schema objects (Appendix B).

7.3.5 Retrieval Procedures

In the implementation of DATAM on PRIMDAS, each instance set corresponds to a single value set. Retrieval is therefore available only in terms of individual instances. This means that to retrieve data as concatenations of instances, procedures need to be constructed. Examples involving attributes are described in this section. The examples also illustrate the relationship of the schema structure to retrieval.

As described in Chapter 6 and Sec. 7.3.2, a composite attribute does not have its own value set, but is instead defined on the instance sets of its component attributes. The retrieval of an instance of a composite attribute requires the retrieval of each component of that instance. Since the retrieval would typically be specified in terms of the composite attribute, access of the schema structure is required to determine its component attributes. This is achieved by using GETLNK through the link lCONTN (Fig. 7.3) which represents composition among attributes. As the composition may be of many levels, this access is nested. Following this, each attribute instance can be retrieved and concatenated into some workspace.

This procedure is extended to retrieve the instances of attributes in general, since it is possible to determine from the schema structure, whether an attribute is composite or not.

Another retrieval involving concatenation is that of the

retrieval of all the attributes of a single object - i.e. the traditional record or n-tuple. This retrieval would typically be specified in terms of the object whose attribute instances are to be obtained. Therefore, the schema structure is again used here, to determine the set of attributes of the object.

Code for the above retrievals is listed in Figs C.19 to C.21. In general, the storage of structural information in the schema enables minimisation in the information that need be specified in retrievals.

7.4 ENTERPRISE MODEL CONSTRUCTION ON DATAM

Functions of the DATAM schema structure (Fig. 7.3) and their usage by procedures representing DATAM operators are described in Sec. 7.3. A current implementation of DATAM, consisting of a primitive interface to such procedures, is described in Appendix C. This section describes an example of the construction and use of an enterprise model and data base with DATAM in terms of this implementation.

7.4.1 Enterprise Model Construction

The enterprise modelled is a subset of a university environment, the DATAM diagram of which is given in Fig. 7.6. Objects of interest are Students, Courses, Staff, Projects and relationships and events among them. The PRIMDAS configuration corresponding to the DATAM diagram is given in Fig. 7.7. Construction of this PRIMDAS configuration and the entry of its DATAM descriptions into the DATAM schema are invoked by DATAM range and create procedures, some of which are described in Secs 7.3.1 and 7.3.2.

The command strings, input to DATAM to create the Student data base structures, are given in Fig. 7.8. This DATAM implementation accepts a two character code to indicate the procedure to be invoked, followed by data for the procedure. A list of codes and their functions is given

in Fig. C.1.

For example, the code "CR" represents the range declaration procedure CRANGE (Sec. 7.3.1). The string "<<CREDIT-PNT><A><2>>" following the code specifies the entry of a new range called CREDIT-PNT with type alpha and length 2 characters into the schema. Similarly, "<<YEAR-STUDY><I>>" enters the range YEAR-STUDY of type integer. "CE" invokes the routine CRENTY (Appendix C), which creates a PRIMDAS structure for the instances of a new entity and appropriately updating the DATAM schema. The string, "<<CAR><CAR-REG>>", for instance, declares an entity CAR whose tokens are of the range CAR-REG.

Sub-entities are declared by "CS" so that <<ACD-STAFF><STAFF>> specifies an entry into the DATAM schema that ACD-STAFF is a sub-entity of STAFF. CRATTR, which enters attribute specifications is invoked by the code "CA". The string "<<COURSE><CREDIT-PNT><CREDIT-PNT><N:1>>" input to CRATTR, describes the attribute CREDIT-PNT of the object COURSE to be created. The range of the attribute is also called CREDIT-PNT, while the attribute association is declared to be N:1.

The other commands SC, CL and CV of Fig. 7.8 invoke routines to enter composite attributes, relationships and events, respectively. The first routine SETCMPO is described in Sec. 7.3 while the others are described in Appendix C.

7.4.2 Data Entry

The next step in the data base construction is the entry of the actual data. This is invoked by the command strings in Fig. 7.9.

The entry routines described in Sec. 7.3 ensure that any cardinality of association is not violated. They are suitable for the entry of single instances, such as those which occur in the casual update of the data base. For large amounts of data, this involves time consuming checking which can be avoided if it is assumed that the data

has been vetted beforehand by some routine suitable for large input. This is assumed in the routines used in Fig. 7.9 which load data directly into the structures.

In Fig. 7.9, the "NA" command loads entity-attribute associations, while "SA" and "VE" commands load subentity-attribute and event-attribute associations, respectively. The other command, "RU", loads relationships. The routines and command string format associated with these commands are described in Appendix C. A general feature of these commands is the last parameter, which indicates the file in which the data for that command is available.

Fig. 7.10 gives sample input files for the commands. For example, the file STUDENTDATA is used by the first command in Fig. 7.9. The first record specifies that Student "29" has Initial "MM", Surname "ROBERTSON", Address "54 SPRINGFIELD RD" and Course-type "F". The file ENROLLDATA contains entries for the event Enrolment. Its first record represents that Student "29" is enrolled in Course "GEOG303" for which the Student has Repeat-indic "N" and Grade "B-".

The files STDCARDATA and SUPRVSNDATA contain entries for the relationships Std-car and Std-suprvsn respectively. For instance, from the first RU command in Fig. 7.9, it is seen that the first record of file STDCARDATA specifies that Student "51" is related through Std-car to Car "DU3711".

7.4.3 Data Base Manipulation

To illustrate the use of the other procedures of Sec. 7.3, Fig. 7.11 gives examples of commands to retrieve, delete and destroy objects of the data base.

Command (1) with code GA invokes GTATTR, to retrieve the composite attribute Name of Student "51". For the data of Fig. 7.10, the output is "CL, BRYANT". GTALLA is invoked by the second command,

which retrieves an attribute record of Student "32":

"SJ, DONOVAN, 331 GREERS RD, F"

where "," are item delimiters.

The third and fourth commands invoke the delete procedures DLOBJ and DLATTR, respectively. The "DO" command deletes Student "29", causing the deletion of all its associated data, which from Fig. 7.10 are:

1. attributes "MM", "ROBERTSON", "54 SPRINGFIELD RD", "F".
2. events in Enrolment, where Student "29" is associated to Courses "GEOG303" and "COSC202", as well as their attributes.
3. the relationship in Std-suprvsn, where Student "29" is associated with Acd-staff "LOY".

In contrast, the "DA" command merely deletes the Address attribute "428 OXFORD TCE" of Student "64".

Finally, commands (5), (6) invoke the destroy procedures DSATTR and DSOBJ, respectively. Command (5) destroys the attribute Course-type which is an attribute of Student. No other actions are involved. On the other hand, command (6) which invokes the destroy of the entity Staff has far reaching consequences. All attributes and any associations in which Staff is involved become destroyed, as are any sub-entities of Staff. Consider the execution of this command on the DATAM diagram and corresponding PRIMDAS configuration in Figs 7.6 and 7.7. The resultant DATAM diagram and PRIMDAS configuration are given in Figs 7.12 and 7.13, respectively. This action corresponds to the complete removal of Staff from the data base.

7.5 EVOLUTION PROCEDURES

Appendix C includes the PRIMDAS procedures of two DATAM evolution operations: one concerned with the progression of an attribute to an

entity and the other to evolve a relationship to an event. Sec. 6.3.4 describes approaches in the PRIMDAS representation of these operations. This section augments these descriptions by referring to the implemented procedures and indicating the usage of the DATAM schema structure.

7.5.1 Attribute → Entity

In the evolution of an attribute, say B, to an entity, a link structure is created between the PRIMDAS structure of the attribute and that of the object it describes. All the associations available through their direct associations must be entered into the link. As the instance representations of attributes and entities are different, this requires appropriate rearrangement of the items of B. In the MAKENTY routine of the ATTRTOENTY procedure (Fig. C.22), this is done by creating a new entity structure, FTEMP, TRANSFERing appropriate values into it from B, destroying B, and finally renaming FTEMP to B.

In the evolution, other attributes in the same conceptual containment hierarchy are also evolved into entities. The tree traversal is performed by the GODOWN routine. This routine accesses the lCONTN link of the DATAM schema structure which represents the associations of conceptual containment.

Other schema structure updates, such as those to record the change of the attribute to an entity are taken care of by making use of other DATAM procedures. In particular, CRENTY and DSATTR are invoked to construct the resultant entity and to destroy the obsoleted attribute, respectively.

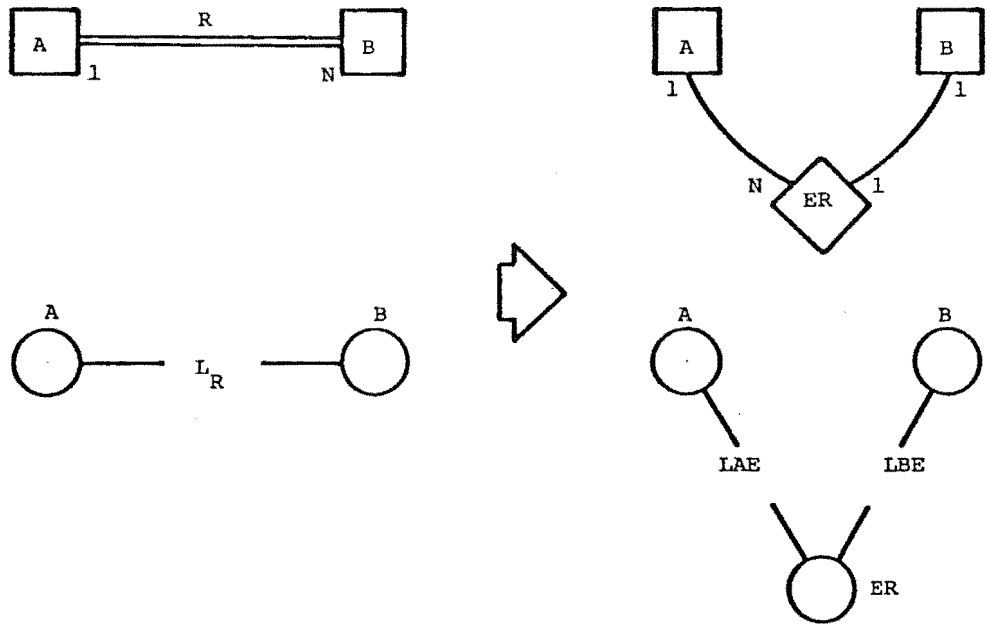
The link created in the operation represents a DATAM relationship and as such must be assigned a name. This name, and names of any other consequent relationships, are input in the command strings to the procedure. For example, the command

#AE<<DEPT><STAFF><STAFF-IS-IN-DEPT>>

invokes the progression of the Dept attribute of Staff in Fig. 7.6 to an entity in its own right. The attribute association then becomes the relationship Staff-is-in-Dept. Since no conceptual containment hierarchy is involved, no further attribute to entity evolution is triggered.

7.5.2 Relationship → Event

The other evolution procedure, RELNTOEVNT, listed in Fig. C.23, changes a relationship to an event. In this implementation, the PRIMDAS links used to represent the involvement of an object in an event is hidden to DATAM users. Therefore, their names have to be generated by the procedure. Two such links of involvement are required in the progression. This is indicated in the diagram below, which illustrates the evolution.



The name generation and link creation is done in the CREVNT procedure which is invoked by RELNTOEVNT. After these links and the token value set (ER in the above diagram) have been created, the PRIMDAS operator SPLIT is used to enter the associations contained in the relationship into the link of involvement. This is followed by the destruction of the link representing the relationship and the renaming of the event

token value set to the given name.

A further consideration in the evolution of a relationship to an event is the splitting of the cardinality of mapping of the association in two for each of the links of involvement. The rules for the splitting are given by the table below:

A:B		A:ER	ER:B
1:1		*1:1*	*1:1*
1:n	⇨	*1:n*	*1:1*
n:1		*1:1*	*n:1*
n:m		*n:m*	*n:m*

Other decompositions are possible, e.g. *n:m* to *n:1* and *1:n*, but further information is required in the choice. The above are arbitrarily selected.

The cardinality of R between A and B is obtained by accessing lRNUM of the DATAM schema structure from lRELN through lRR. Access of the schema structure is also required to determine the objects between which the relationship is defined, since only the relationship name is input to the procedure. As with the ATTRTOENTY procedure, schema updates are also carried out in RELNTOEVNT by invoking other DATAM operators.

An example of the evolution of a relationship to an event on the student data base of Fig. 7.6 is the command:

#RE<<CRS-RSPNSBTY><CRS-RSPNSBTY>>

This invokes the evolution of the relationship Crs-rspnsbty to an event of the same name. This evolution may be motivated by the desire to describe the course responsibility of academic staff by the particular duties that they have to carry out.

7.6 ALTERNATIVE VIEW

This section describes how subviewing can be approached in a PRIMDAS implemented DATAM system. Further, an example of the construction of alternative views is given by considering the formation of a network view of a DATAM subview.

7.6.1 Support of Subviews

The schema structure of Fig. 7.3 contains the complete description of the object and association types contained in the data base. A subview of this community view would require a further schema structure which allows access to only those objects contained in the subview. Such a subschema structure can be most simply facilitated by providing four mark sets, one for each of the DATAM schema structure v/m sets lENTY, lEVENT, lRELN, lATTRI. An example is illustrated in Fig. 7.14, where the subschema structure objects are prefixed by "ll" to identify the particular subview.

These subschema structures would contain only those objects deemed to be in the subview. For example, consider the subview of the student data base, given in Fig. 7.15. The subschema structure object, llENTY, for this subview would contain the entities

Course, Student, Acd-Staff .

Similarly for llEVENT, llRELN and llATTRI which contain the events, relationships and attributes of the subview.

Since these subschema objects are mark sets of the community schema objects, all the associated DATAM descriptions existing in the schema structure of Fig. 7.3 are available from the four subschema objects of Fig. 7.14. Simple modifications allowing for the subschema structures would enable the data base procedures of Sec. 7.3 to be usable for subviews. The entry of data into the subschema objects can be

effected by procedures which mark objects into the four structures based on some subview declaration.

7.6.2 Network View

The support of an alternative view, such as a network view, requires a further schema structure to contain the description of the subview under consideration in terms of the alternative model. The PRIMDAS configuration of such a schema structure for a network view of a DATAM data base on PRIMDAS is given in Fig. 7.16.

The value set RECORD-TYPE contains the names of the record-types of the view, while OWNED-RECORD-TYPE is a mark set containing those record-types which are owned. Set-types, which correspond to PRIMDAS links, are contained in the mark set SET-TYPE. The associations of record-types through the set-types are represented by the links SWITH, STO, HASS and the value set M-N-LINK.

The fields of the record types correspond to either value sets or mark sets. In the transformation of DATAM objects to network structures, non-key fields correspond only to attributes. This is indicated by the network schema object, NON-KEY FIELD, which is a mark set of the DATAM schema object, lATTRI. Key fields, on the other hand, correspond to token instance sets, which could be either a value set or a mark set. This is reflected in the network schema structure by the object KEYFIELD being a mark set of OVMSET.

Based on the correspondence described in Chapter 4, the network configuration corresponding to the DATAM subview of Fig. 7.15 is given in Fig. 7.17. Further structures need to be added to the PRIMDAS configuration of the DATAM view to allow for the network representation of relationships (Sec. 6.4.2).

Fig. 7.18 gives the resultant PRIMDAS configuration, where objects prefixed with a "2" are those included for the above reason.

For example, the composition of the links 2HASPREREQ and 2ISPREREQ is derived from and evolves with, the original link PREREQ. That is, the value sets 2PREREQ and 2CRS-RSPNSBTY are dummy ones to facilitate network navigation.

Based on the above considerations, following is an algorithm to form a network view of a DATAM configuration. Its implementation involves accessing the appropriate DATAM subschema structure and inserting appropriate entries into the schema structure of the network view.

```
procedure CONSTRUCT-NETWORK-VIEW-OF-DATAM-SUBVIEW
begin
     for each describable object
          enter the object name into RECORD-TYPE;
               link it to the value/mark set name of the object,
                    which is entered in KEYFIELD;
                    link it to the value set names of the object's
                         attributes which are entered in NON-KEY
                         FIELD;
     for each event
          enter its name into OWNED-RECORD-TYPE;
          record its key field and non-key fields as for describable
               objects above;
          record the associations between the event, each involved
               member and the involved link into STO, M-N-LINK,
               SWITH, HASS and SET-TYPE;
     for each relationship
          partition its link structure using a dummy value set
               which represents the key of a dummy record-type;
          record this record-type, its links etc. as for events
               above;
end;
```

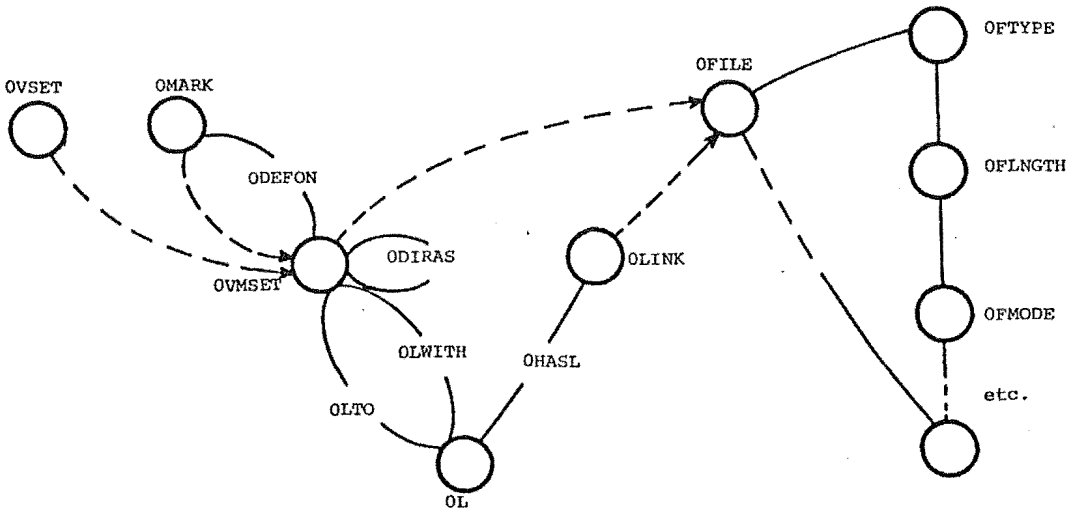


FIGURE 7.1 PRIMDAS configuration of PRIMDAS schema structure

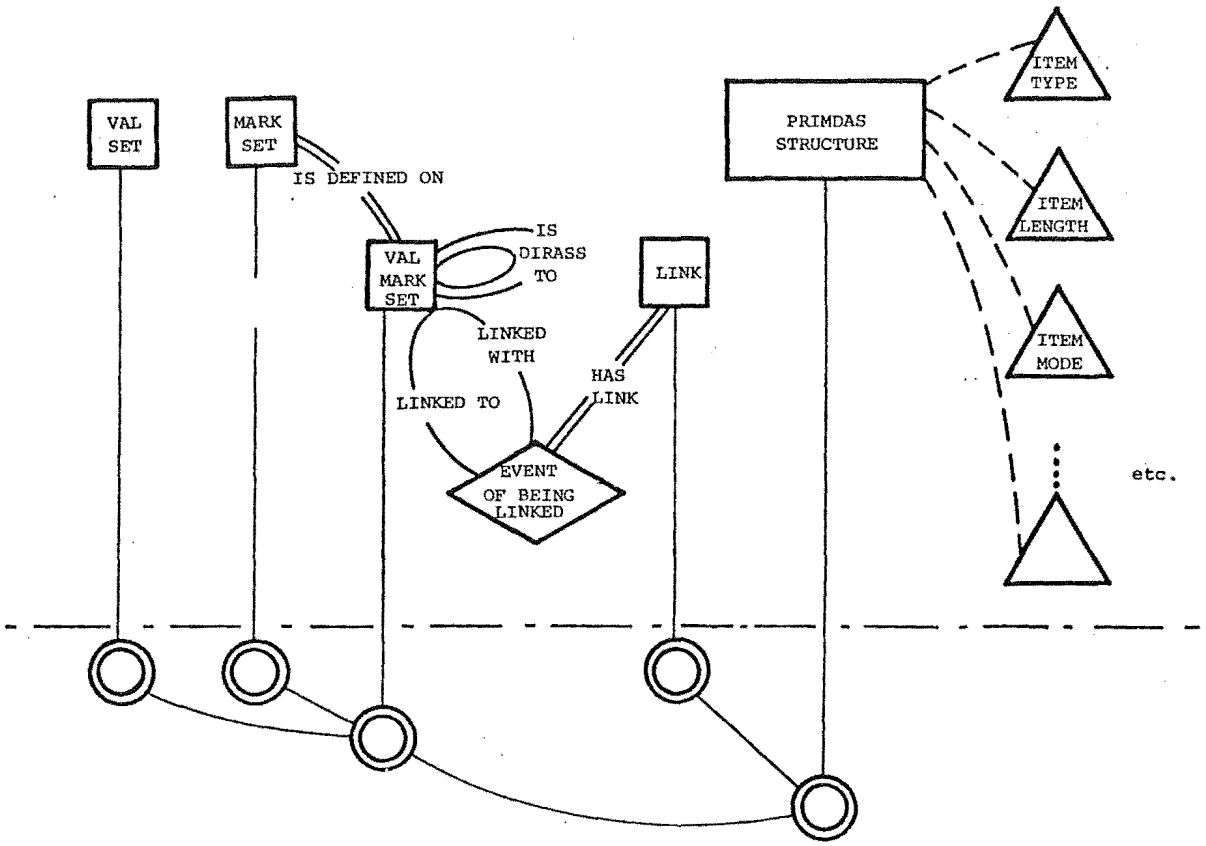


FIGURE 7.2 DATAM diagram of PRIMDAS schema structure

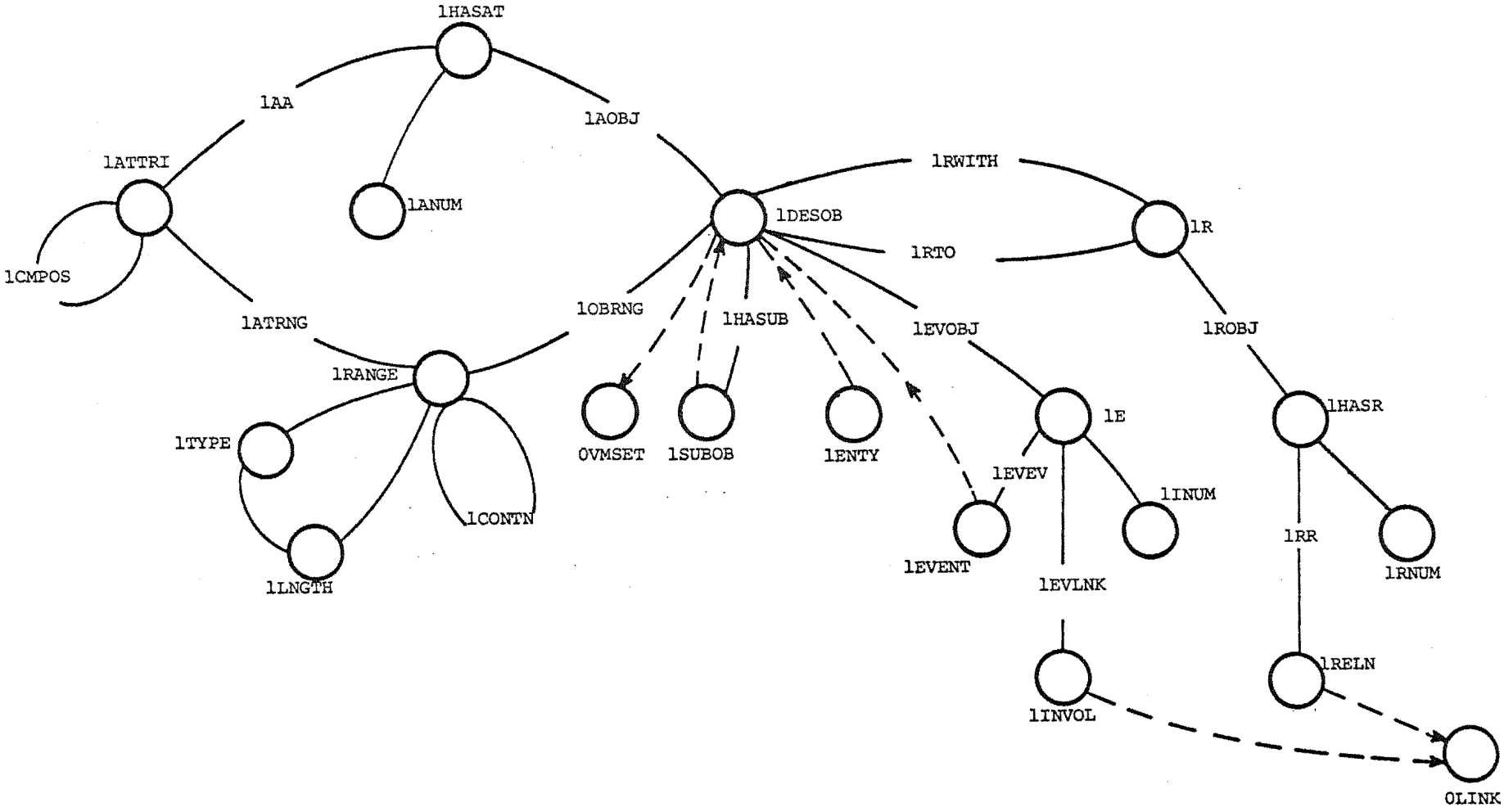


FIGURE 7.3 PRIMDAS configuration of DATAM schema structure

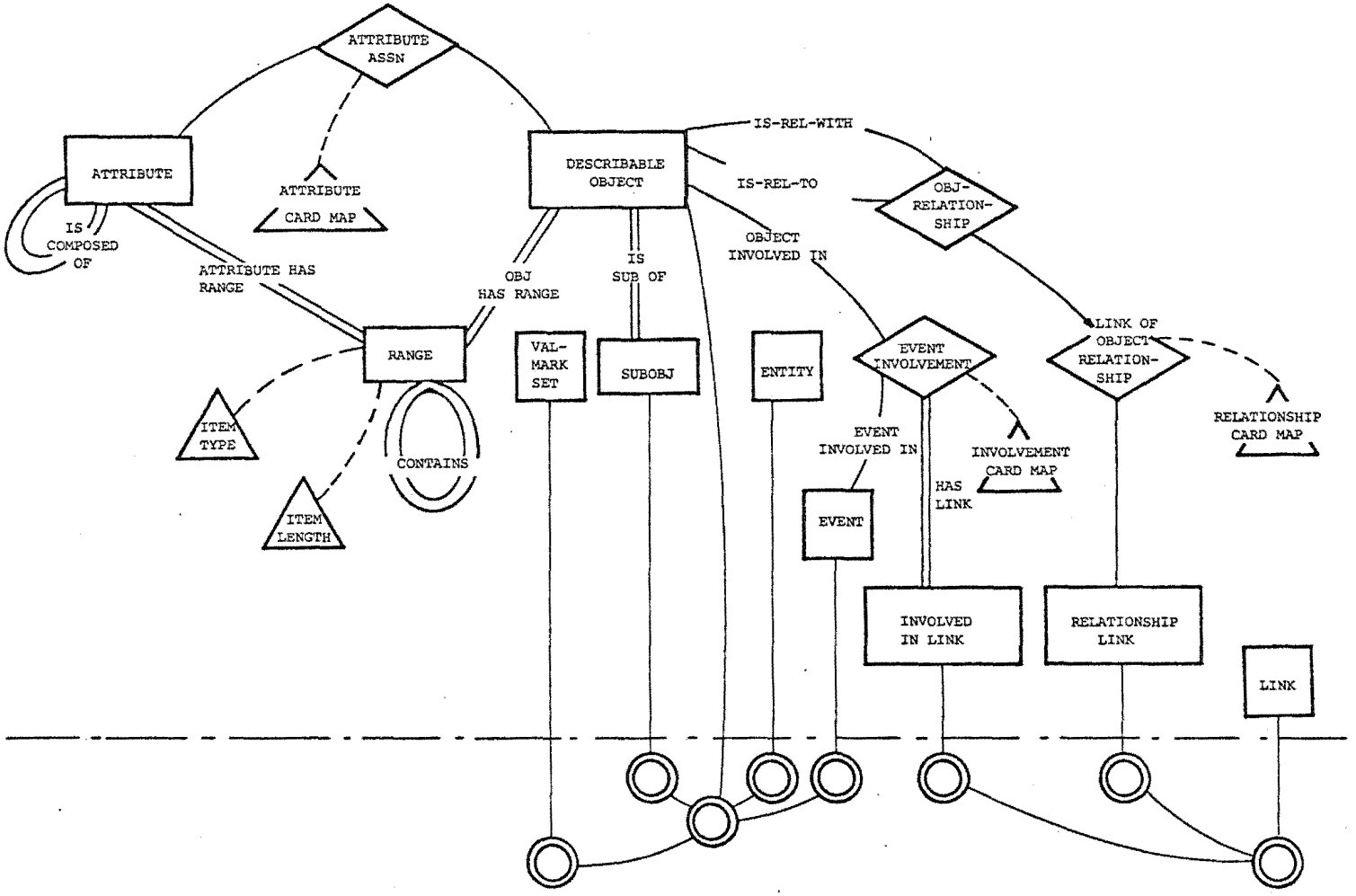


FIGURE 7.4 DATAM diagram of DATAM schema structure

```
CR#V#1RANGE#A#2#
CR#V#1TYPE#A#1#
CR#V#1LNGTH#I#
CR#V#1R#I#
CR#V#1E#I#
CR#V#1ANUM#A#1#
CR#V#1INUM#A#1#
CR#V#1RNUM#A#1#
CR#M#1DESOB#OVMSET#
CR#M#1SUBOB#1DESOB#
CR#M#1ENTY#1DESOB#
CR#M#1EVENT#1DESOB#
CR#V#1ATTRI#A#2#
CR#M#1INVOL#OLINK#
CR#M#1RELN#OLINK#
CR#L#1RTO#1DESOB#1R#
CR#L#1HASUB#1DESOB#1SUBOB#
CR#L#1RWITH#1DESOB#1R#
CR#L#1EVOBJ#1DESOB#1E#
CR#L#1EVEV#1EVENT#1E#
CR#L#1EVLNK#1E#1INVOL#
CR#V#1HASAT#I#
CR#V#1HASR#I#
CR#L#1ROBJ#1R#1HASR#
CR#L#1RR#1RELN#1HASR#
CR#L#1AOBJ#1DESOB#1HASAT#
CR#L#1AA#1ATTRI#1HASAT#
CR#L#1ATRNG#1ATTRI#1RANGE#
CR#L#1OBRNG#1DESOB#1RANGE#
CR#L#1CONTN#1RANGE#1RANGE#
CR#L#1CMPOS#1ATTRI#1ATTRI#
DI#1HASAT#1ANUM##
DI#1HASR#1RNUM##
DI#1E#1INUM##
DI#1RANGE#1TYPE#1LNGTH##
```

FIGURE 7.5 Commands to construct DATAM schema structure

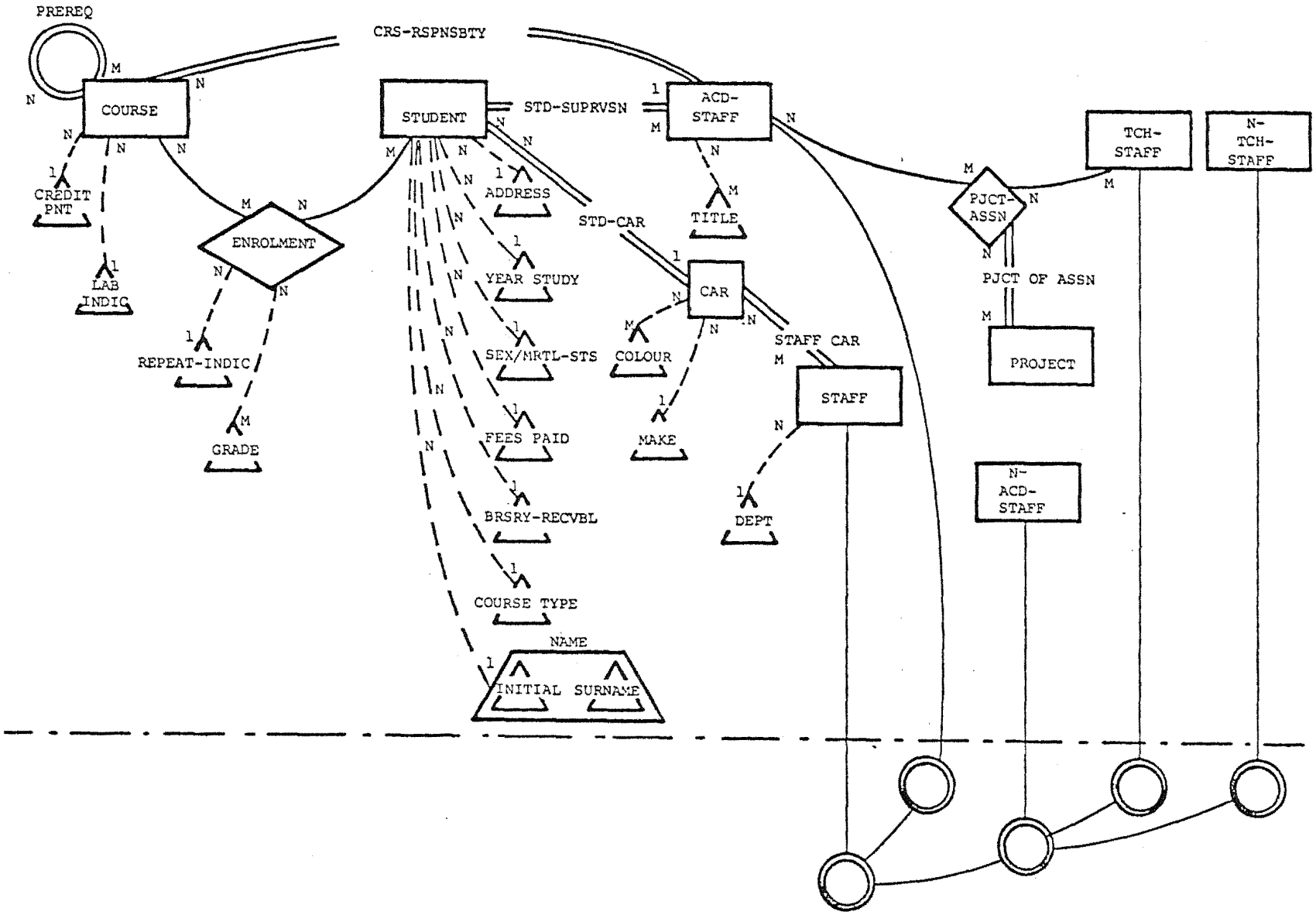


FIGURE 7.6 DATAM diagram of university data base

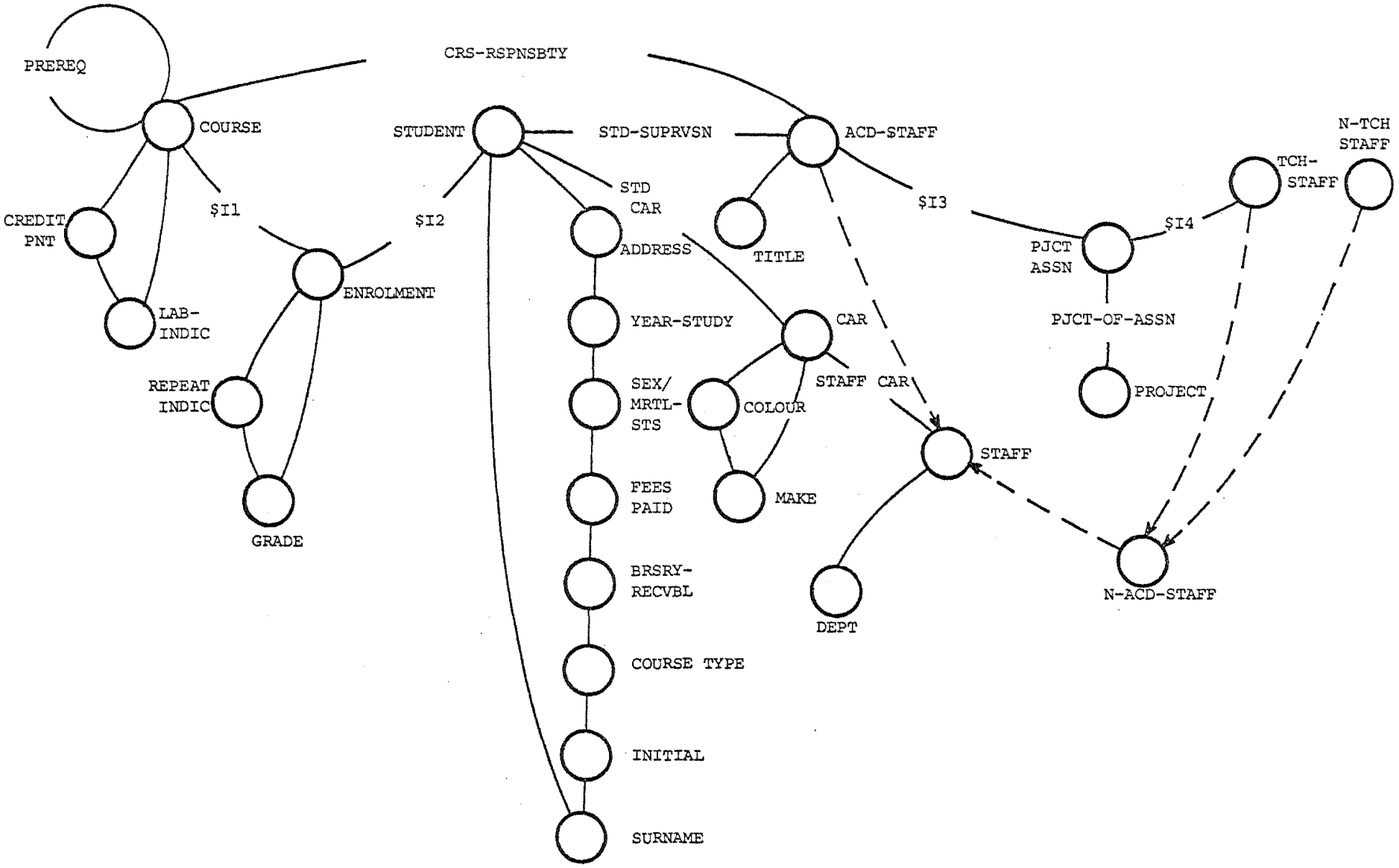


FIGURE 7.7 PRIMDAS configuration of university data base

```
#CR

<<COURSE-CODE><A><7>>
<<CREDIT-PNT><A><2>>
<<LAB-INDIC><A><1>>
<<STUDENT-ID ><A><4>>
<<ADDRESS><A><30>>
<<YEAR-STUDY><I>>
<<SEX/MRTL-CDE><A><1>>
<<FEES-PAID><R>>
<<BRSRY-RECVBL><R>>
<<COURSE-TYPE><A><1>>
<<INITIAL><A><3>>
<<SURNAME><A><20>>
<<CAR-REG#><A><6>>
<<COLOUR><A><15>>
<<CARMAKE><A><15>>
<<STAFF-NAME><A><25>>
<<DEPT><A><20>>
<<TITLE><A><4>>
<<PROJECT-NAME><A><10>>
<<REPEAT-INDIC><A><1>>
<<GRADE><A><3>>

#CE

<<COURSE><COURSE-CODE>>
<<STUDENT><STUDENT-ID >>
<<CAR><CAR-REG>>
<<STAFF><STAFF-NAME>>
<<PROJECT><PROJECT-NAME>>

#CS

<<ACD-STAFF><STAFF>>
<<N-ACD-STAFF><STAFF>>
<<TCH-STAFF><N-ACD-STAFF>>
<<N-TCH-STAFF><N-ACD-STAFF>>

#CA

<<COURSE><CREDIT-PNT><CREDIT-PNT><N!1>>
<<COURSE><LAB-INDIC><LAB-INDIC><N!1>>
<<STUDENT><ADDRESS><ADDRESS><N!1>>
<<STUDENT><YEAR-STUDY><YEAR-STUDY><N!1>>
<<STUDENT><SEX/MRTL-STS><SEX/MRTL-CDE><N!1>>
<<STUDENT><FEES-PAID><FEES-PAID><N!1>>
<<STUDENT><BRSRY-RECVBL><BRSRY-RECVBL><N!1>>
<<STUDENT><COURSE-TYPE><COURSE-TYPE><N!1>>
<<STUDENT><INITIAL><INITIAL><N!1>>
<<STUDENT><SURNAME><SURNAME><N!1>>
<<CAR><COLOUR><COLOUR><N!N>>
<<CAR><MAKE><CAR-MAKE><N!1>>
<<STAFF><DEPT><DEPT><N!1>>
<<ACD-STAFF><TITLE><TITLE><N!1>>

#SC

<<STUDENT><NAME><N!1><<INITIAL><SURNAME>>>

#CV

<<ENROLLMENT><><<<COURSE><N!N>><<STUDENT><N!N>>>>
<<PJCT-ASSN><><<<ACD-STAFF><N!N>><<TCH-STAFF><N!N>>>>

#CA

<<CAR><MAKE><CARMAKE><N!1>>
<<ENROLLMENT><REPEAT-INDIC><REPEAT-INDIC><N!1>>
<<ENROLLMENT><GRADE><GRADE><N!N>>

#CL

<<STD-SUPRVSN><STUDENT><ACD-STAFF><N!N>>
<<CRS-RSPNSBTY><COURSE><ACD-STAFF><N!1>>
<<PREREQ><COURSE><COURSE><N!N>>
<<STD-CAR><STUDENT><CAR><N!1>>
<<STAFF-CAR><STAFF><CAR><N!N>>
<<PJCT-OF-ASSN><PJCT-ASSN><PROJECT><N!N>>
```

FIGURE 7.8 DATAM commands to construct university data base.

```
#NA %ENTITY-ATTRIBUTE LOAD

<<STUDENT><<<INITIAL><SURNAME><ADDRESS><COURSE-TYPE>><<DISK><STUDENTDATA>>>>
<<COURSE><<<CREDIT-PNT><LAB-INDIC>><<CARD><COURSEDATA>>>>
<<CAR><<<COLOUR><MAKE>><<CARD><CARDATA>>>>
<<STAFF><<<DEPT>><<CARD><STAFFDATA>>>>
<<PROJECT><<><<CARD><PROJECTDATA>>>>

#SA %SUB-OBJECT-ATTRIBUTE LOAD

<<ACD-STAFF><<<TITLE>><<CARD><ACDSTAFFDATA>>>>
<<N-ACD-STAFF><<><<CARD><NACDSTAFFDATA>>>>
<<TCH-STAFF><<><<CARD><TCHSTAFFDATA>>>>
<<N-TCH-STAFF><<><<CARD><NTCHSTAFFDATA>>>>

#VE %EVENT-ATTRIBUTE LOAD

<<PJCT-ASSN><<ACD-STAFF><TCH-STAFF>><<><<<CARD><PJCTASSNEVNTDATA>>>>
<<ENROLLMENT><<COURSE><STUDENT>><<<REPEAT-INDIC><GRADE>><<<DISK><ENROLLDATA>>>>

#RU %RELATIONSHIP LOAD

<<STD-CAR><STUDENT><CAR><<DISK><STDCARDATA>>>
<<PREREQ><COURSE><COURSE><<CARD><PREREQDATA>>>
<<CRS-RSPNSBTY><COURSE><ACD-STAFF><<CARD><RSPNSBTYDATA>>>
<<STAFF-CAR><STAFF><CAR><<CARD><STAFFCARDATA>>>
<<STD-SUPRVSN><STUDENT><ACD-STAFF><<<DISK><SUPRVSNDATA>>>

#TH
<<PJCT-OF-ASSN><<PJCT-ASSN><<ACD-STAFF><TCH-STAFF>>><PROJECT><<<CARD><PJASDATA>>>
```

FIGURE 7.9 DATAM commands to load data.

```
  DATA STUDENTDATA

<<29><<MM><ROBERTSON><54 SPRINGFIELD RD><F>>
<<51><<CL><BRYANT><230 CLYDE RD><F>>
<<64><<W><FURLONG><428 OXFORD TCE><P>>
<<32><<SJ><DONOVAN><331 GREERS RD><F>>

  DATA ENROLLDATA

<<<GEOG303><29>><<N><B->>>
<<<COSC202><29>><<N><A>>>

  DATA STDCARDATA

<<51><DU3711>>
<<32><DJ3234>>

  DATA SUPRVSNDATA

<<32><RIDWAN>>
<<29><LOY>>
```

FIGURE 7.10 Sample input data.

```
#GA <<NAME><51><STUDENT>>
#GL <<32><STUDENT>>
#DA <<64><><STUDENT><ADDRESS>>
#DO <<29><STUDENT>>
#XA <COURSE-TYPE>
#XO <STAFF>
```

FIGURE 7.11 Sample DATAM manipulation commands.

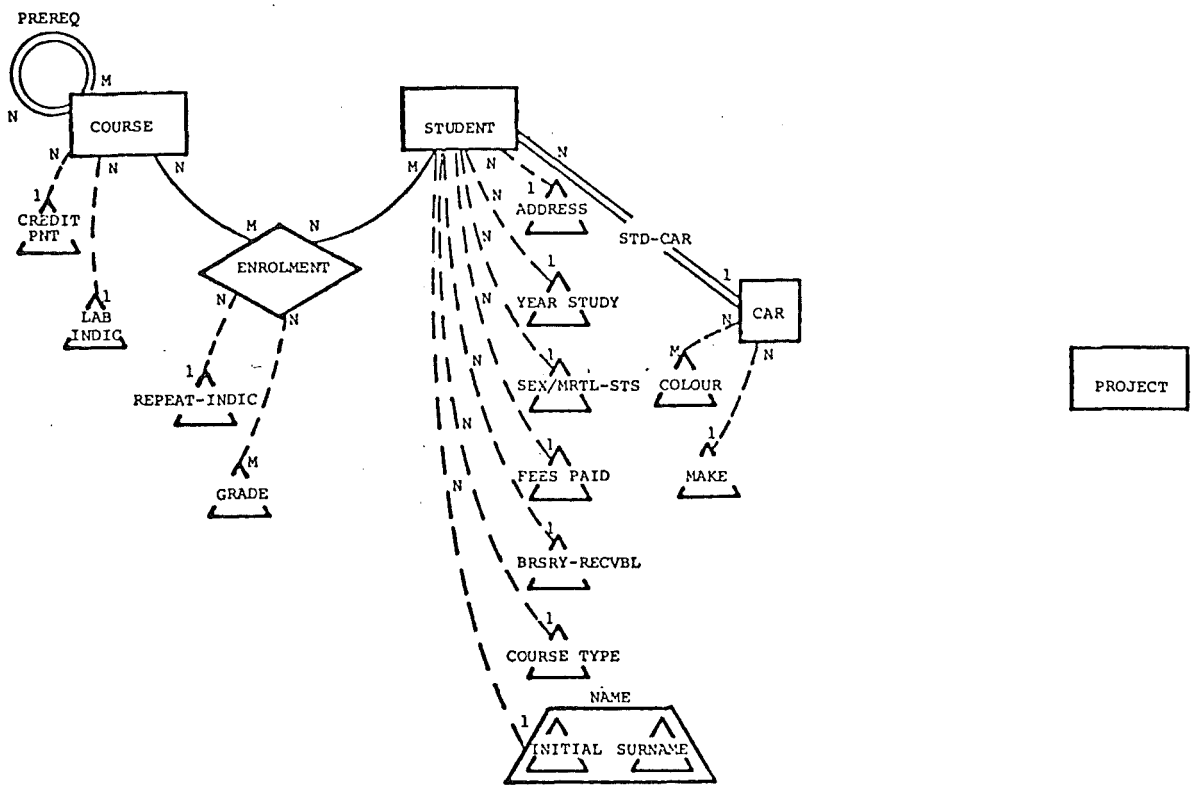


FIGURE 7.12 DATAM diagram after #XO<STAFF>

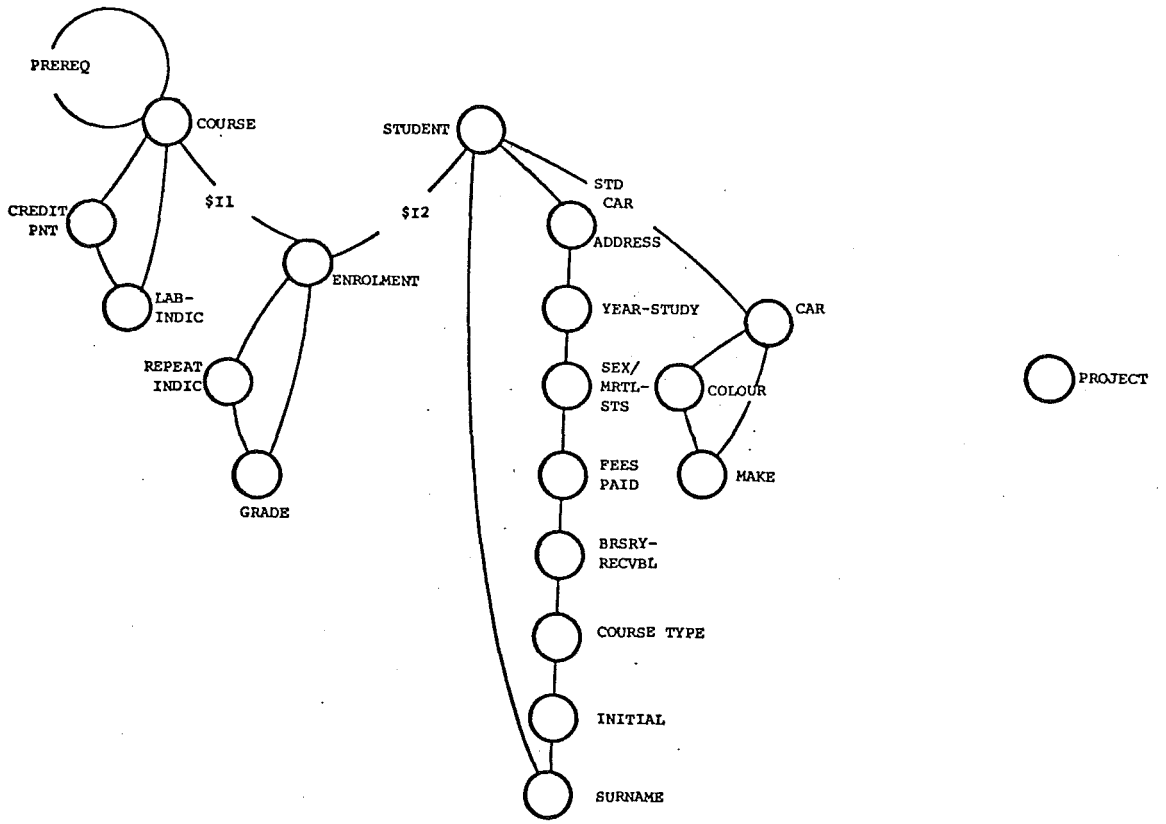


FIGURE 7.13 PRIMDAS configuration after #XO<STAFF>

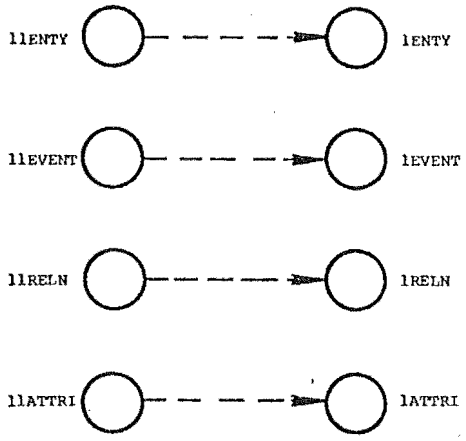


FIGURE 7.14 Subschema structure

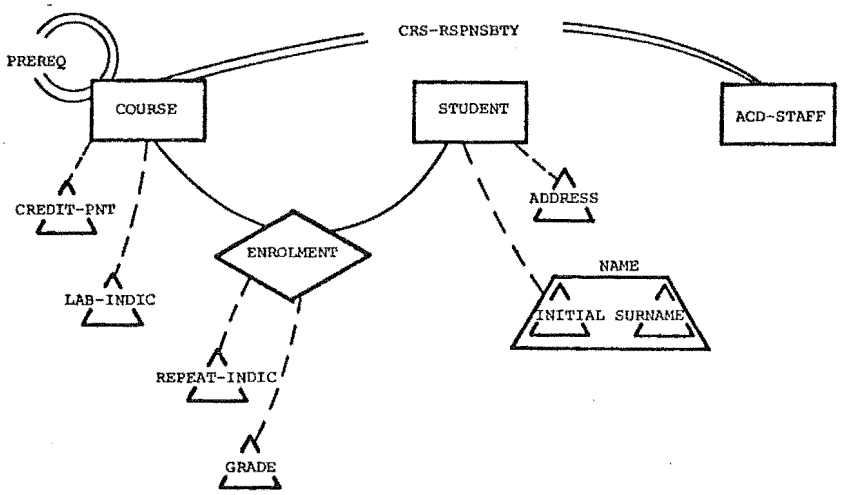


FIGURE 7.15 Subview of student data base

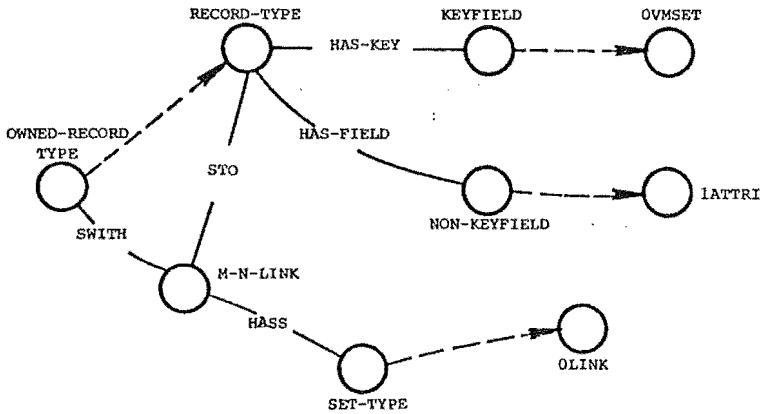


FIGURE 7.16 Network schema structure

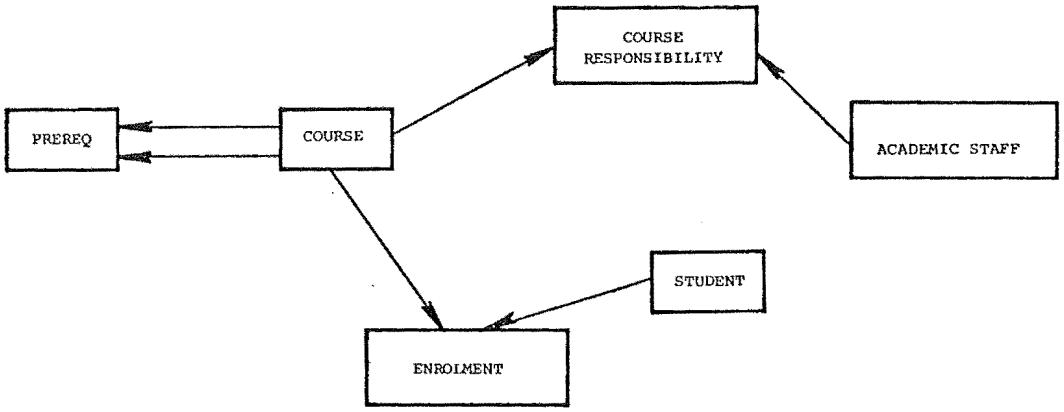


FIGURE 7.17 Network view of DATAM subview of Figure 7.15

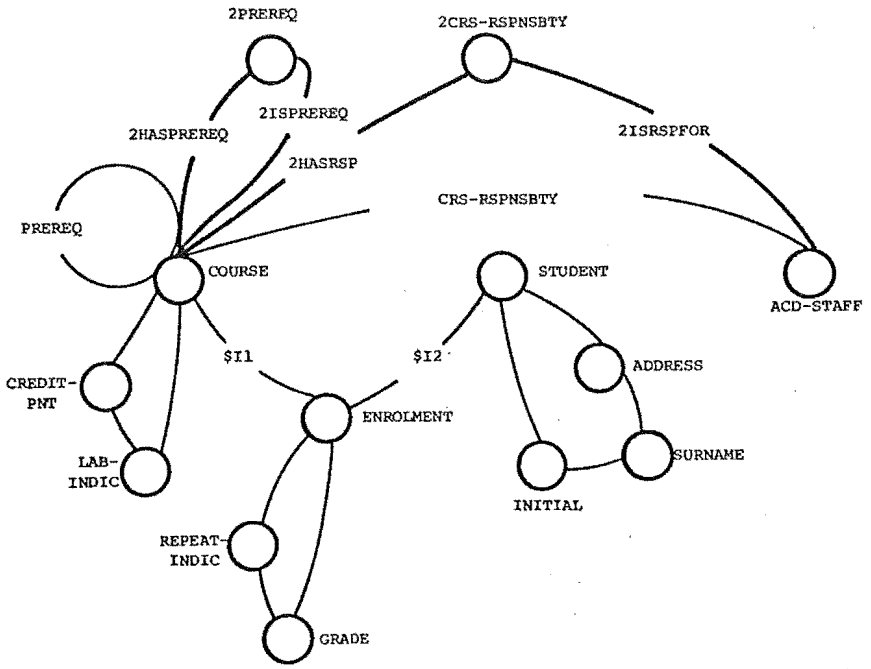


FIGURE 7.18 PRIMDAS configuration to support network view

CHAPTER 8

CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH

8.1 CONCLUSION

This thesis considers issues in data base systems: the gross architectural framework and the abstractions at two major levels (conceptual data base modelling and data base model support) in this framework.

Multi-level architectural specifications in the literature confuse the relationships of three distinct data base system functional components: data representation, data perception and data base languages. In Chapter 2, the roles and interactions of multiple level representations of these three functions are clearly formalized. This enables goals and criteria in architectural construction to be more specifically determined.

Further, Chapter 2 distinguishes, in architectural terms, between the independent support of multiple models and coexistence. This is of value in exposing architectural considerations that are critical to each type. The distinction also indicates points in the architecture at which they can be mutually supported.

Criticisms of architectural specifications of the literature extend to their diagrammatic representations. An alternative is presented in Chapter 2, which indicates more precisely the inter-relationships among software levels. The diagrams provide immediate visual appraisals useful in the gross comparisons of systems.

In coexistence, two levels of abstraction are significant: the data base modelling level and the level at which different models

are to be supported. This thesis investigates these levels intensively, presenting in each case a new model.

Chapter 3 includes a comprehensive treatment of abstractions in the modelling of the real world. A conceptual framework is developed, providing a unifying approach in the treatment of real world abstractions. Constructs evident in the literature are encompassed and placed in perspective. Extensions and refinements are introduced. While comprehensive, the framework and the diagrammatic representations emphasize simplicity, a key criteria for community models.

In particular applications, not all abstractions are utilized. The provision of a range of data base systems of varying modelling power is advocated. A spectrum giving a graduation of conceptual complexity for data base systems is introduced.

Chapter 4 introduces a new data base model, DATAM, which is based on the framework developed in Chapter 3. DATAM fills a gap at the evolutionary point in the conceptual spectrum. An extensive set of operations to evolve the data base, required to complement such a system, is described. Further, examples in the correspondence of DATAM to other abstract models and the formation of subviews are given. The simplicity of DATAM objects and their precise relationship to concepts of the real world make these functions unambiguous and easily understood. These factors, as well as the examples of DATAM as a semantic reference for data models, support the suitability of DATAM as a community model.

Simplicity of concept is also adopted in PRIMDAS, presented in Chapter 5. PRIMDAS, an implementation base for the support of data base models, contains generalized concepts, including those of data associational constructs, embedded consistency and restructuring operators. A low level procedural data sublanguage is chosen to allow a wider range of applicability. Its item rather than record-based data

structuring enhances its flexibility. PRIMDAS represents a generalized implementation base suitable for multiple model support.

Chapter 6 establishes the role of PRIMDAS in the representation of data base models. PRIMDAS is seen to reduce the effort required in developing new models, since the distance to the machine is shortened. It is possible to adopt a structurally uniform approach in which any subsidiary data base (e.g. schema-handling) requirements as well as temporary structures in processing, all use PRIMDAS. PRIMDAS is shown to be able to support DATAM directly. The structural support of other models - e.g. binary, network - are also immediate.

The support of coexistence on PRIMDAS is illustrated by examples. The particular case of network viewing of DATAM requires little effort. In any case, representations on PRIMDAS can be chosen to simplify alternative viewing.

The feasibility of PRIMDAS is demonstrated by its implementation. Construction of a primitive DATAM system on a current PRIMDAS implementation is described in Chapter 7 and supported by code listings in the Appendices. Extension of the DATAM implementation to provide subviewing and alternative viewing is outlined.

8.2 SUGGESTIONS FOR FURTHER RESEARCH

This thesis has contributed to an understanding in data base system architecture, modelling and support. This study can be used as the basis for constructive symbolic formalisms (Kraegeloh and Lockemann [1977], Maibaum [1977]) which can be used for the derivation of meaningful extensions and results.

The implementation and application of PRIMDAS and DATAM are currently undergoing study in the construction of an educational data base system at the University of Canterbury. DATAM and PRIMDAS will be

used in this system to develop interfaces to various models and to study practical factors in the coexistence architecture. In this, considerations such as efficiency, concurrency and security will require further research.

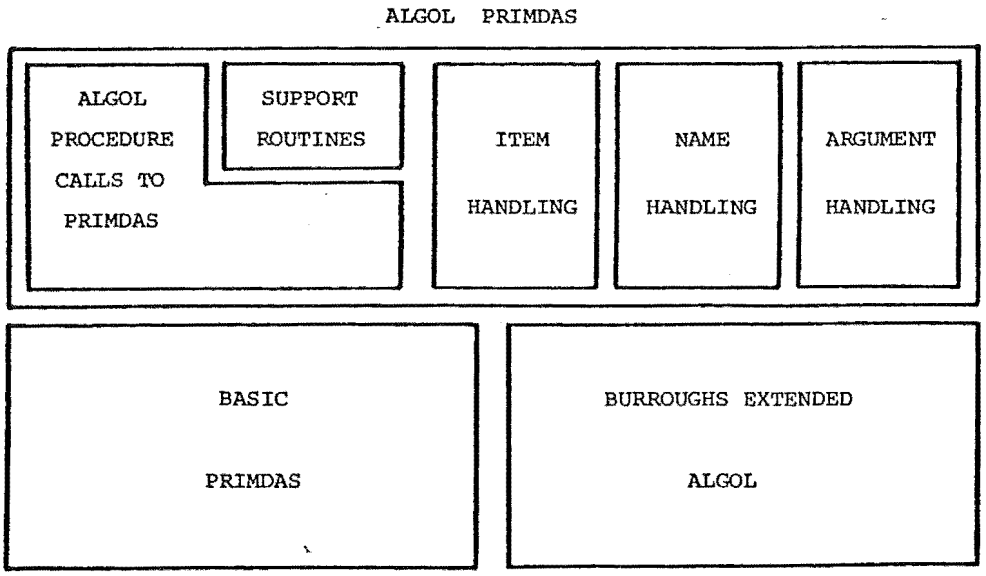
In conceptual data modelling, simplicity, ease of formulation and understanding are considered important criteria. These are currently unquantifiable. However, as with other subjective criteria, experiments can be used to obtain relative measures in performance (Reisner *et al*. [1975], Lochovsky [1977], Lochovsky and Tsichritzis [1977]). The design and application of such experiments would be of great value in providing a facility for evaluating conceptual frameworks such as that presented in this thesis.

APPENDIX A

A PRIMDAS INTERFACE

A.1 OVERVIEW

This appendix describes details of the PRIMDAS system interface as currently implemented on a Burroughs B6700 at the University of Canterbury. A schematic representation of the interface components is given below.



The Basic PRIMDAS component represents the actual storage and retrieval system. Sec. A.2 describes the operators currently provided by Basic PRIMDAS and how communication with it is carried out.

Algol PRIMDAS is the form of PRIMDAS as embedded in Burroughs Extended Algol (Burroughs Corporation [1977]). Its components are described in Secs A.3 and A.4. Sec. A.3 discusses item, name and argument handling facilities, while Sec. A.4 describes the form of the Algol procedural calls representing the PRIMDAS operators. Sec. A.4

also includes support routines which represent often used PRIMDAS actions as pseudo operators.

A.2 BASIC PRIMDAS

Commands and their parameters are passed to PRIMDAS as character strings. The character "#" is used as argument delimiters. The first argument is a two letter code representing the PRIMDAS operator to be invoked, followed by its parameters in the appropriate format. Fig. A.1 gives the operators currently supported, its two letter codes and the form of its parameters. Examples of create command strings are given in Fig. 7.5, while command strings of other operators are given in Fig. A.2. The commands in Fig. A.2 represent calls to PRIMDAS generated by the CRANGE (Sec. 7.3) routine in declaring a range to DATAM.

Basic PRIMDAS returns a status code containing a number representing the operator invoked, and a code indicating the error status on exit from PRIMDAS. If no error occurred, a zero code is returned. The numbers corresponding to PRIMDAS procedures and the possible error codes are given in Fig. A.3. This scheme follows the guidelines of the April 71 CODASYL DBTG Report (CODASYL [1971]). A Boolean variable, e.g. SUCCESS, is typically defined in the sublanguage to correspond to the success or failure of an operation.

PRIMDAS makes a requested item available in a PRIMDAS workspace, RESULT. The item is provided in its workable form - viz., real, integer or character - and is in a format that facilitates item handling. This is discussed in Sec. A.3.

A Boolean routine, EOF, tests the status of specific cursors. For example, EOF(SA) is true if the cursor SA is currently not at an item in the set on which it is defined.

A.3 PROGRAMMING AIDS

This section describes the facilities provided to ease the use of PRIMDAS in its procedural form. The facilities are classified into item, name and argument handling.

A.3.1 Item Handling

The item handling facilities involve formatting Algol arrays to allow a uniform treatment in some common item operations. These formatted arrays are called DAVARS (Data Variables) on which the following operations are defined:

(1) MOVE (A, B) moves the contents of Davar A to Davar B.

(2) COMP (A, B, flag) compares Davar A and Davar B in terms of the item types. Flag is used to indicate the relational condition to be tested. The conditions NE, LT, etc. can be defined on this operator.

(3) PRNT (A) prints out a DAVAR's Contents in some standard format.

(4) MAKARG (A, buf) reformats the DAVAR's Contents into buf in a form suitable for input to PRIMDAS.

(5) PUT (buf, A, mode, length) formats the string in buf into Davar A in terms of the given mode and length.

(6) PUTSAME (B, buf, A) is defined as
PUT (buf, A, mode (B), length (B)), whose effect is to format buf into A in terms of the mode and length of the contents of Davar B.

Items retrieved from PRIMDAS are made available as DAVARs.

A.3.2 Name Handling

Provision for unique names is often necessary when using named searches and temporary PRIMDAS structures. A facility is included which allocates to DATAM separate pools of search names and PRIMDAS structure

names. Names can be obtained from these pools when required, and returned when finished.

To use this facility in the routines, the following steps are followed (the corresponding conventions for PRIMDAS structure names are given in brackets):

(1) Declare an area in which to contain pointers to the search names [structure names]. This is done by

SEARCHNAMES (area, #reqd) [FILENAMES (area, #reqd)]

(2) Obtain the names by the operation

GETSNAMES (area, #reqd) [GETFNAMES (area, #reqd)]

After this the names become available as

SNAMES [area[i]] [FNAMES [area[i]]]

$0 \leq i < \#reqd$

where SNAMES and FNAMES contain the name strings.

(3) On exit from the routine, the names are returned by

RETSCH (area, #reqd) [RETFIL (area, #reqd)]

Examples can be seen in the code listed in Appendix C.

A.3.3 Argument Handling

Arguments are passed to DATAM enclosed in angled brackets. Nested bracketing is used to allow for a variable number of arguments and to enable arguments to have components. Since the routines have differing argument formats, unpacking is done individually for each routine. The facilities take the form of Burroughs Algol pointer setting operators. Following are those used in the DATAM model construction:

(1) SETONARG (argp, buf) sets the pointer argp to the complete argument string in buf.

(2) SETARGOF (subargp, argp) sets subargp to the first component of the arg pointed to by argp.

(3) SETARGAFTER (aftarg, argp) sets aftarg to the arg after argp at the same level.

(4) For variable number of arguments, the operation

NEXTARG (argp) sets argp to the next argument following argp at the same level and sets SUCCESS appropriately,

NOMOREARG (argp) is a boolean procedure which invokes NEXTARG and is true if it fails, and

RESETARG (argp) sets argp to the first arg at the same level as the initial argp.

(5) EXIST (argp) is a boolean which is used to test whether a particular argument is entered.

(6) ARGLENGTH (argp) returns the character length of argument argp.

A.4 ALGOL EMBEDDED PRIMDAS

A set of procedures representing PRIMDAS operators is built on top of Basic PRIMDAS. Parameters to these procedures are pointers to character strings, which are then used by the procedures to form the appropriate command strings to Basic PRIMDAS. The form of the procedure calls follows closely that described in Chapters 5 and 6. Fig. A.4 gives the format as used in the code listed in these appendices.

Algol PRIMDAS also includes further procedures taking the role of pseudo PRIMDAS operators. These operators are not primitive and are defined in terms of the procedures in Fig. A.4. The operators and their code are given in Fig. A.5. Each is briefly described below:

(1) NEXTU (CURSOR) and NEXTM (CURSOR) are cursor setting operators based on the ordering of value or mark sets. The first sets CURSOR to the next item in the set which is different from that currently pointed to by CURSOR. NEXTM sets CURSOR to the next item, but returns SUCCESS as true only if that item is the same

as that at which CURSOR is initially.

(2) MEMBER (VALUE, VMSET) is a boolean procedure which is true if and only if VALUE is an item of VMSET.

(3) The COMLNK procedure generates a new instance for an Integer value set and links it to items in two other associated value sets. This operator is used when entering events involving two objects, where the tokens are default so that identification is always through the involved objects.

(4) The procedure OBJHASATTR accesses directly associated items to determine whether a given item is directly associated to any other in a specified set. In DATAM terms it serves to determine whether a given token has an attribute of the given type.

(5) While the previous operations are generally applicable, the procedure MKEINVNAME specifically concerns the DATAM schema model object, lINVOL. lINVOL contains the names of the links used to represent the involvement of objects in a db-event. These names (prefixed by $I) are invisible to a DATAM user and are generated by MKEINVNAME. Item generation such as MKEINVNAME and COMLNK are based on the sets being ordered so that the current largest value can be known.

SYNTAX OF PRIMDAS COMMANDS

BASIC ELEMENTS:

```
<Search>::= ##|<searchname>
<Searchname>::= {nonempty string not containing a "#" and of length ≤6 characters}#
<Filename>::= <VMset name>|<link name>
<VMset name>::= <valueset name>|<markset name>
<Valueset name>::= <file identifier>#
<Link name>::= <file identifier>#
<Markset name>::= <file identifier>#
<File identifier>::= {nonempty string not containing a "#" and of length ≤12 characters}
<VMset list>::= <VMset name><VMset list>|<VMset name>#
<Optional VMset list>::= <VMset list>|##
<Searchname list>::= <searchname><searchname list>|<searchname>#
<Optional searchname list>::= <searchname list>|##
<Link association list>::= <link association><link association list>|<link association>#
<Optional link association list>::= <link association list>|##
<Link association>::= <linkname>B#|<linkname>F#|<linkname>##
<Common link list>::= <common link association><new common link list>
<New common link list>::= <common link association><new common link list>|<common link association>#
<Common link association>::= <link association><search>
<Source part>::= F#@|F#<file title>#|F#DISK#<file title>|P#@|P#<searchname list>|
          A#<optional searchname list><instance list>
<DBsetting>::= <search>|ASSOC#<optional link association list><search><VMset name>|
          AT#<common link list><VMset name>
<Database name>::= <database identifier>#
<Database identifier>::= {valid directory name for the Burroughs I/O subsystem}
<Filetitle>::= {valid file title in Burroughs I/O subsystem, followed by "."}
<Instance list>::= <instance><instance list>|<instance>@
<Instance>::= {string ended by "#"}|##
<Optional search>::= <search>|@
<Optional integer string>::= #|<integer string>#
<Integer string>::= {string of decimal characters}
```

COMMANDS:

```
<APPEND>::= AP#<optional VMset list><VMset list><search><source part>
<ASSOC>::= AS#<optional link association list><search><VMset name>
<CREATE>::= CR#V#<valueset name>A#<integer string>#|
          CR#V#<valueset name>I#|
          CR#V#<valueset name>R#|
          CR#M#<markset name>#<VMset name>|
          CR#L#<link name>#<VMset name><VMset name>
<CREATE INDEX>::= CI#<valueset name>
<CREATE MODEL>::= CM#<database name>
<DELETE>::= DE#<DBsetting>
<DELINK>::= DL#<link name><search><search>
<DEMARK>::= DM#<DBsetting>
<DESTROY>::= DS#<filename>
<DEASSOCIATE>::= DA#<filename>
<DIR ASSN.>::= DI#<VMsetname><VMset list>
<EMPTY>::= EM#<markset name>
<ENTER>::= EN#<VMset list><source part>
<ENDS>::= ED<search>
<FIRST>::= FI#<search>
<GET>::= GE#<DBsetting>
<GETLINKS>::= GL#<optional link association list><search><VMset name><markset name><optional integer string>
<LAST>::= LA#<search>
<LINK>::= LI#<link name><search><search>
<MATCH>::= MA#<search><instance>
<MARK>::= MR#<DBsetting><markset name>
<NEXT>::= NE#<search>
<PRIOR>::= PR#<search>
<RENAME>::= RN#<filename><filename>
<REPLACE>::= RE#<DBsetting><instance>
<SEARCH>::= SE#<VMset name><search>
<SIGN ON>::= SN#<data base name>
<SETAT>::= ST#<DBsetting><search>
<TRANSFER>::= TR#<DBsetting><valueset name><optional search>
<SPLIT>::= SP#<linkname><linkname><linkname>
<PARTITION>::= PT#<linkname><linkname><linkname>
<MERGE>::= ME#<linkname><linkname><linkname>
<UNION>::= UN#<linkname><linkname><linkname>
```

FIGURE A.1 Syntax and code of PRIMDAS commands

DATAM command:

```
#CR<<COURSE-CODE><A><7>>
```

Generated PRIMDAS calls:

```
SE#1RANGE##
MA##COURSE-CODE#
ED##
SE#1RANGE#1SO1#
EN#1RANGE##A#1SO1##COURSE-CODE##@
AP##1TYPE##1SO1#A##A##@
AP##1LNGTH##1SO1#A##7##@
ED#1SO1#
```

FIGURE A.2 Calls to PRIMDAS for CRANGE

(a)

PRIMDAS COMMANDS

	COMMAND NAME	COMMAND NUMBER		COMMAND NAME	COMMAND NUMBER
1	APPEND	24	17	GETLINKS	101
2	ASSOC	26	18	LAST	113
3	CREATE	41	19	LINK	116
4	CREATE INDEX	36	20	MATCH	129
5	CREATE MODEL	38	21	MERGE	130
6	DELETE	50	22	MARK	137
7	DELINK	53	23	NEXT	146
8	DEMARK	54	24	PARTITION	235
9	DESTROY	58	25	PRIOR	233
10	DEASSOCIATE	49	26	REPLACE	162
11	DIRECT ASSN.	52	27	SEARCH	178
12	EMPTY	70	28	SET AT	187
13	ENDS	64	29	SIGNON	183
14	ENTER	71	30	SPLIT	184
15	FIRST	84	31	TRANSFER	201
16	GET	98	32	UNION	247

(b)

ERROR NO.	MEANING	ERROR NO.	MEANING	ERROR NO.	MEANING
1	Searchname not defined	34		67	
2	Search at EOF	35		68	Too many searchnames - max 10
3	No records in v/m set	36		69	
4	No entries in link structure	37	Common link not found	70	Missing B or F for links
5	Already signed on	38		71	Missing <source part> i.e. F, P or A
6		39		72	
7	Multiple search	40	File not found or is PRIMDAS schema object	73	
8		41	Mark attempt to PRIMDAS schema mark set	74	
9		42		75	Missing <file type> i.e. V, M or L
10	Unsuccessful match	43	Update attempt on PRIMDAS schema value set	76	Missing <file mode> i.e. A, R or I
11	Item with this index already exists	44		77	Data base name not given
12		45		78	Data base already present
13	Input record in error-too long	46		79	File name cannot start with 0
14	Input record in error-invalid type	47		80	File name already exists
15	Input file not found	48		81	File type not mark set
16	Wrong mark set association	49		82	
17		50	Search not found	83	
18	Wrong link association	51		84	
19		52		85	No association for DELINK
20	Wrong direct association	53	v/m set is read only	86	
21		54	v/m set is not EMPTY	87	
22		55		88	
23	Unsuccessful ASSOC	56		89	
24		57	Structure name too long	90	Disk file title not followed by "."
25	GETLNK - none found	58		91	Stack search not present
26	GETLNK - required no. not found	59		92	Two search updates for same file
27		60	Missing "#"	93	Search not defined on any entering files
28		61	Double hash not expected	94	Data base setting unsuccessful
29		62	Not enough link structures	95	Invalid record length
30	No link structures given	63	Too many structure names - max 10	96	Search already defined
31	No v/m set names given	64	Not enough v/m set names given	97	Process or IO time exceeded
32		65	Mark set level up to 5 only	98	Serious error - contact supervisor
33	Search not defined on this v/m set	66		99	Illegal command

FIGURE A.3 PRIMDAS command and error numbers

(a) Command numbers
(b) Error numbers

```
APPEND    ([VRP], VAP, cursor, input items)
          VRP - value sets of items to be duplicated
          VAP - input value sets
ASSOC     ([LINKS], cursor, target v/m set)
          LINKS - sequence of link structures
AT        (ATCS, target v/m set)
          ATCS - sequence of <link, cursor> pairs
CREATE    (TYP, name, VMD, VLN)
          TYP - one of {VSET, MSET, LSET}
          (i)     when TYP = VSET:
                        VMD - item mode
                        VLN - item length
          (ii)    when TYP = MSET:
                        VMD - source v/m set
                        VLN - not used
          (iii)   when TYP = LSET:
                        VMD - v/m set being linked
                        VLN - v/m set being linked
DELETE    (<dbsetting args>)
          <dbsetting args>::= STYP, SCS, ASCUR, target v/m set.
          STYP - one of {AT, ASSC}
          (i)     when STYP = AT:
                        SCS - sequence of <link, cursor> pairs
                        ASCUR - not used
          (ii)    when STYP = ASSC:
                        SCS - sequence of links
                        ASCUR - cursor
DELINK    (link, source-cursor, dest-cursor)
DEMARK    (<dbsetting args>)
DESTROY   (structure name)
DIRASS    (v/m set 1, v/m set 2)
          note:  associates pairs of v/m sets
EMPTY     (v/m set)
ENDS      ([cursor])
ENTER     (v/m set, item, [cursor])
          note:  only single item entry
FIRST     ([cursor])
GET       (<dbsetting args>)
GETLINK   (LINKS, cursor, v/m set, mark set)
          note:  only ASSC access
LAST      ([cursor])
LINK      (link, source-cursor, dest-cursor)
MARK      (<dbsetting args>, mark set)
MATCH     ([cursor], value)
NEXT      ([cursor])
PRIOR     ([cursor])
RENAME    (old name, new name)
SEARCH    (v/m set, [cursor])
SETAT     (<dbsetting args>, target cursor)
SPLIT     (source-link, link 1, link 2)
TRANSFER  (<dbsetting args>, target subset, target cursor)
```

Notes:

(1) Square brackets indicate optional parameters.
In Appendix C, the pointer "D" is used to indicate a null argument.

(2) The function of operators in this figure and in Figure A.1 not described in the text (e.g. DESTROY, RENAME) is indicated by its name.

FIGURE A.4 Algol PRIMDAS call formats

```
COMMENT   SETS CURSOR TO NEXT
          UNIQUE ITEM;

PROCEDURE NEXTU(SNAME); %<SEARCHNAME>
VALUE SNAME; POINTER SNAME;
BEGIN
  REAL ARRAY TEMP[0:50];

  GET(D,D,SNAME,D);
  MOVE(RESULT,TEMP);
  DO BEGIN
    NEXT(SNAME)
  END
  UNTIL NE(RESULT,TEMP)OR NOT SUCCESS;

END;

COMMENT   SETS CURSOR TO NEXT ITEM,
          SUCCESS:=TRUE IF ITEM SAME
          AS PREVIOUS;

PROCEDURE NEXTM(SNAME); %<SEARCHNAME>
VALUE SNAME; POINTER SNAME;
BEGIN
  REAL ARRAY TEMP[0:50];

  GET(D,D,SNAME,D);
  MOVE(RESULT,TEMP);
  NEXT(SNAME);
  IF NE(RESULT,TEMP) AND SUCCESS
  THEN SUCCESS:=FALSE

END;

COMMENT   TRUE IF VALUE IS IN THE
          GIVEN VAL/MARK SET;

BOOLEAN PROCEDURE MEMBER(VALU,VMSET); %<VALUE,VAL/MARK SET>
VALUE VALU,VMSET; POINTER VALU,VMSET;
BEGIN

  SEARCH(VMSET,STSRCH);
  MATCH(STSRCH,VALU);
  IF SUCCESS THEN
  MEMBER:=TRUE
  ELSE
  MEMBER:=FALSE;
  ENDS(STSRCH);

END;


COMMENT   GENERATE NEW NAME FOR 1INVOL
          HAS PREFIX "&I";

PROCEDURE MKEINVNAME(NEWNAME);
EBCDIC ARRAY NEWNAME[0];
BEGIN INTEGER NAMENUM;

  SEARCH(INVOL,STSRCH);
  REPLACE NEWNAME[0] BY "&I" FOR 2;
  LAST(STSRCH);
  IF NOT SUCCESS THEN NAMENUM:=0
  ELSE
  NAMENUM:=INTEGER(POINTER(RESULT[3])+2,4)+1;
  REPLACE NEWNAME[2] BY NAMENUM FOR 4 DIGITS,
    "#" FOR 1;
  ENDS(STSRCH)

END;
```

```
COMMENT   GENERATE NEW EVENT TOKEN,
          INPUT:  TV=TARGET VALUE SET
                  TC=CURSOR ON TV(SET TO NEW TOKEN)
                  L1,C1,L2,C2=TWO PAIRS OF SOURCE
                              LINKS AND CURSORS;

PROCEDURE COMLNK(TV,TC,L1,C1,L2,C2);
VALUE TV,TC,L1,C1,L2,C2;
POINTER TV,TC,L1,C1,L2,C2;
BEGIN EBCDIC ARRAY NXTVAL[0:50];

  REPLACE NXTVAL[0] BY ATARG(L1,C1,L2,C2);
  SETAT(AT,NXTVAL,D,TV,TC);
  IF NOT SUCCESS THEN BEGIN
  LAST(TC);
  IF NOT SUCCESS THEN REPLACE NXTVAL BY "0##"
  ELSE BEGIN
  RESULT[3]:=++1;
  MAKARG(RESULT,NXTVAL);
  END;

  ENTER(TV,NXTVAL,TC);
  LINK(L1,C1,TC);
  LINK(L2,C2,TC)

  END
END;



COMMENT   TRUE IF TOKEN OF GIVEN OBJECT TYPE
          HAS ASSOCIATED INSTANCE IN THE
          SPECIFIED ATTRIBUTE;

BOOLEAN PROCEDURE OBJHASATTR(OBJ,TOKEN,ATTRI);
VALUE OBJ,TOKEN,ATTRI;
POINTER OBJ,TOKEN,ATTRI;
BEGIN BOOLEAN HASA;

  HASA:=FALSE;

  SEARCH(OBJ,STSRCH);
  MATCH(STSRCH,TOKEN);
  WHILE NOT HASA AND SUCCESS
  DO BEGIN
    ASSOC(D,STSRCH,ATTRI);
    IF SUCCESS THEN HASA:=TRUE
    ELSE NEXTM(STSRCH)
  END;
  ENDS(STSRCH);

  OBJHASATTR:=HASA
END;
```

FIGURE A.5 Algol PRIMDAS support routines

APPENDIX B

DATAM SCHEMA STRUCTURE

This appendix describes the DATAM schema structure used in the current implementation (Fig. 7.3). The configuration stores information on the DATAM objects existing in the data base and their correspondence to PRIMDAS objects.

The discussion is in sections, describing the representation of each of the concepts of ranges, attributes, entities, relationships and events separately.

B.1 RANGE

The set of range names are stored in the value set lRANGE (Fig. 7.3). The value sets lLNGTH and lTYPE are directly associated to lRANGE and contain the length and type, respectively, of the ranges. Sub and super range associations are represented by the link lCONTN from lRANGE to itself.

B.2 ATTRIBUTE

The value set lATTRI contains all attribute names. The link lATRNG associates attributes to the range names on which they are defined.

Associations of composition among attributes are represented by the link, lCMPOS, from lATTRI to itself.

B.3 ENTITY

The set of describable object names, consisting of all entities, events as well as their sub-objects are stored in lDESOB. This structure is a mark set of the PRIMDAS schema object OVMSET, reflecting that all describable objects correspond to either value sets or mark sets of the same name.

Subsets of describable objects, consisting of all entities, events and sub-objects, are in the mark sets lENTY, lSUBOB and lEVENT respectively. The association of describable objects to their sub-objects is represented by the link lHASUB. The link lOBRNG associates describable objects with the names of the ranges on which their tokens are defined.

Attribute associations are events involving attributes and describable objects. This event is represented by the value set lHASAT which is linked to lATTRI by lAA corresponding to the attribute link of involvement. lAOBJ is the describable object link of involvement in the attribute association. The association of objects to attributes is available through the links lAA and lAOBJ.

The cardinality of attribute associations is contained in the value set lANUM, which is directly associated to lHASAT.

B.4 EVENT

The event of the involvement of objects in db-events is represented by the value set lE. The object of the involvement is available through the link lEVOBJ between lDESOB and lE, while the link to the event is represented by lEVEV. The actual PRIMDAS link structures which provide the involvement are stored in lINVOL, which is a subset of the set of all links, OLINK. Each such link is

associated to its event of involvement in lE through lEVLNK. The cardinality of the involvement of an object in an event is contained in the value set lINUM, which is directly associated to lE.

B.5 RELATIONSHIP

Relationship names correspond to the names of their corresponding links. They are contained in lRELN which is a mark set of the set of all links, 0LINK.

There may be many relationships between any two describable objects. The event that two objects are related is represented by the value set lR, with the links of involvement lRWITH and lRTO. For each such event, its associations to the relationships between the two objects are represented in the value set lHASR. These are described by its cardinality stored in the value set lRNUM. The set of relationships between two objects is available through lROBJ and lRR.

APPENDIX C

DATAM OPERATORS

C.1 OVERVIEW

This appendix describes briefly the PRIMDAS code of the DATAM operators currently implemented. It augments the descriptions given in Chapter 7. Fig. C.1 lists the codes of the operators, the procedures invoked and an indication of their function. These do not exhaust the operators required to provide a complete DATAM system. Extension can be effected by the addition of further procedures as they are required.

The operators are coded using the Algol PRIMDAS described in Appendix A. The bulk of the code is concerned with file and argument handling, which for clarity have been omitted from some of the listings. The code also incorporates routine checking of input, for example to ensure that a given object exists. Any error causes an exit from the procedures. A list of DATAM error numbers and their meaning is given in Fig. C.2(a).

Names of the DATAM schema objects are available in Algol value arrays of the same name, without the prefix "l".

The procedures are described in the following sections. Constructs used in the code and not described in the text are given in Fig. C.2(b).

C.2 RANGE AND OBJECT CONSTRUCTION

There are three range construction routines, CRANGE, SETSUPR and SETSUBR, listed in Fig. C.3. Their functions are described in Sec. 7.3.1.

In CRANGE, the pseudo operator MEMBER is used in checking the uniqueness of input range names. In SETSUPR and SETSUBR, where the LINK is from lRANGE to itself, the direction is determined by the order of the cursors.

Routines involved in DATAM object construction are: CRENTY, CREVNT, CRDESOB, CRSNTY, CRATTR, SETCMPO, ATRLNK and CRRELN, corresponding to the commands CR, CV, CS, CA, SC and CL. These procedures are listed in Figs C.4 to C.7. The construction of entities, sub-objects and attributes is described in Sec. 7.3.2. Routines for the construction of attributes (CRATTR) and composite attributes (SETCMPO) both use the procedure ATRLNK (Fig. C.6(b)) representing actions common to both operations. In ATRLNK, the pseudo operator COMLNK is used to create a unique token for the attribute association event represented in lHASAT.

The following description outlines the procedures for the construction of events (CREVNT) and relationships (CRRELN). The function of the DATAM schema objects used is described in Appendix B.

Steps in the construction of an event corresponding to code in Figs C.4(b) and C.4(c) are given below:

(1) Check that the member objects exist.

(2) Create a value set for the token instance set.
Mark the event name into lEVENT.
Link the event name in lDESOB to its range, if specified.

(3) For each member,

(i) Enter its event of involvement into lE, linking it to lEVENT and lDESOB through lEVEV, lEVOBJ respectively.

(ii) Create its link of involvement (name generated by MKEINVNAME).
Mark this into lINVOL and link it to its event in (i).

(iii) Append the cardinality of involvement (in lINUM) to the event of involvement in (i).

Step (2) has common features to steps in entity construction and is coded in CRDESOB, which is used by both CRENTY and CREVNT.

The construction of relationships is simpler (Fig. C.7) and involves the steps:

(1) Create a link structure between the two input objects.

(2) Enter the event of the association of the two objects into lR.

(3) Mark the relationship link into lRELN.

(4) Enter the event of the association of the relationship to the event of (2) into lHASR. Append to this the cardinality of the relationship.

C.3 DATA ENTRY

The procedures concerned with data entry are NTROBJ, NTRATR, NTRSUB, NTRELN, RELNWITHEVNT and NTREVT, corresponding to the operators EO, LO, NA, EA, LA, SA, RU, TM and VE. Their codes are listed in Figs C.8 to C.12.

The procedures are structured to allow for alternative modes of input. In particular, some routines provide for single entries through the command array, as well as multiple entries from some file. Further forms of input can be catered for as the need arises. The parameter, SOURCE, indicates the particular form being invoked.

Procedures NTROBJ (Fig. C.8) and NTRATR (Fig. C.9) provide for the input of tokens and attributes, respectively. They are described briefly in Sec. 7.3.3. Following are descriptions of the other routines NTRSUB, NTRELN, RELNWITHEVNT and NTREVT.

NTRSUB (Fig. C.10) caters for local enters into sub-objects. This involves marking the token from the source object into the sub-

object. The procedure requires only the input of the token and the sub-object. Its source object is retrieved by access through lHASUB from lSUBOB to lDESOB. This particular program also allows the input of the attributes of the sub-object, without any checks being made on any cardinality considerations.

NTRELN (Fig. C.11(a)) associates pairs of tokens through a given relationship. The algorithm for the procedure is similar to that given in Subsec. 6.3.3.2.3. The routine checks whether the specified relationship exists and is defined between the two given objects, but does not check for violations of the cardinality of the relationship.

The procedure RELNWITHEVNT caters specifically for relationships involving an event and where the entries are specified in terms of the objects involved in the event. The tokens of these objects are used to determine the event token, which is then linked to the other involved object.

The final enter procedure, NTREVT (Fig. C.12), enters events in terms of the tokens of the involved objects. The process involves accessing lINVOL through lEVLNK from lE to obtain the links of involvement of each member object. Cursors are set at each involved token, which are then used to link these tokens to the event token. If it does not already exist, this event token is generated by the procedure. NTREVT also appends attributes of the event.

C.4 DESTROY AND DELETE

Procedures to destroy object-types are DSOBJ, DSATTR and DSRELN, while those to delete instances are DLENTY, DLOBJ, DLATTR and DLEVNT. These are listed in Figs C.13 to C.18.

DSOBJ and DSATTR destroy describable objects and attributes, respectively. They are listed in Figs C.13, C.15 and are described

in Sec. 7.3.4. The other destroy procedure, DSRELN (Fig. C.14), destroys relationships. It deletes the cardinality of the relationship in lRNUM and the event of the relationship in lHASR. The PRIMDAS link structure representing the relationship is then destroyed.

Procedures involved in deleting objects are DLENTY, DLOBJ, DLATTR and DLEVNT. DLENTY (Fig. C.16(b)) deletes entities globally. This means that the deletion of a sub-object is identical to that of a source object, which involves the deletion of the object from the data base rather than just from a subset. The process of the deletion is carried out by the routine DLOBJ (Fig. C.16(a)). This routine deletes any attribute instances, sub-objects and events associated with the object being deleted.

DLEVNT (Fig. C.18), a procedure to delete events, also uses DLOBJ. The event to be deleted is specified by the tokens of its involved objects. DLEVNT uses these to determine the event token which is then input to DLOBJ.

The deletion of attribute instances is invoked by DLATTR (Fig. C.17). This routine deletes either a specified attribute of an object or if not specified, its chronologically first associated instance in the given attribute type.

C.5 RETRIEVAL

GTATTR, GTALLA and PGETVAL are examples of routines for retrieval. GTATTR (Fig. C.20) returns an attribute of a given object. The routine PGETVAL (Fig. C.19) is used to obtain the attribute instance, which may be the concatenation of a number of instances if a composite attribute is required.

PGETVAL is also invoked by the routine GTALLA (Fig. C.21) which retrieves an instance of each of the attribute types associated with a

given object. The links lAOBJ and lAA of the DATAM schema are used to obtain the attribute types associated to the object type. The instances are grouped into the composite attributes of which they may be members.

C.6 EVOLUTION

Two examples of evolution procedures are described in Sec. 7.5. One evolves attributes to entities, and the other evolves relationships to events. These correspond to the routines ATTRTOENTY (Fig. C.22) and RELNTOEVNT (Fig. C.23), respectively.

The algorithm for the schema traversal involved in determining all the attributes to be evolved is:

1. find the range of the given attribute.
2. find the super range of the range.
3. all attributes defined on this range are evolved to entities.
4. follow the link lCONTN down to find all the sub-ranges of the super-range. For each range, all the attributes defined on it are evolved to sub-entities.

The sub-procedure MAKENTY, which performs the restructuring, uses other DATAM procedures. In particular, CRRELN, CRENTY and DSATTR are invoked to construct the resultant relationship and entity and destroy the evolved attribute.

RELNTOEVNT also calls other DATAM operators. CREVNT and DSRELN are used to create the resultant event and to destroy the relationship being evolved.

OPERATOR CODE	FUNCTION	PROCEDURES INVOKED
CA	Construct an attribute	CRATTR, ATRLNK
CE	Construct an entity-type	CRENTY, CRDESOB
CL	Construct a relationship	CRRELN
CR	Enter a range description into the schema structure	CRANGE
CS	Construct a sub-entity	CRSNTY
CV	Construct an event	CREVNT, CRDESOB
DA	Delete attribute instance	DLATTR
DE	Global token delete	DLENTY, DLOBJ
DV	Delete event in terms of member tokens	DLEVNT, DLOBJ
EA	Enter attribute instances with cardinality check	NTRATR
EO	Enter single token	NTROBJ
GA	Retrieve a single attribute instance	GTATTR, PGETVAL
GL	Retrieve an instance of all attributes of an object	GTALLA, PGETVAL
LA	Enter attribute instances with no cardinality check	NTRATR
LO	Enter file of tokens	NTROBJ
NA	Direct entry of token-attribute associations	NTROBJ
RA	Attribute → entity evolution	ATTRTOENTY
RR	Relationship → event evolution	RELNTOEVNT
RU	Direct entry of relationships	NTRELN
SA	Direct entry of subobject-attribute associations	NTRSUB
SB	Associate a range to its contained ranges	SETSUBR
SC	Construct a composite attribute	SETCMPO, ATRLNK
SP	Associate a range to its super range	SETSUPR
TM	Enter relationship involving event in terms of its involved objects	RELNWITHEVNT
VE	Direct entry of events, with respect to member tokens	NTREVT
XA	Destroy attribute	DSATTR
XO	Destroy describable object	DSOBJ
XR	Destroy relationship	DSRELN

FIGURE C.1 DATAM operators

(a)

ERROR NO.	DESCRIPTION
5	Name not unique
10	Range not found
15	Object construction not successful
20	Source object not found
25	Object not found
30	Relationship not found
35	Attribute-type not found
40	Token not unique
45	Cardinality of association violated
50	Relationship and objects do not match
55	Token not found
60	Sub-object not found
65	Missing range
70	Missing object
75	Attribute instance not found
80	Insufficient relationship names provided
85	No event for given tokens
90	Event not found
95	Missing token

FIGURE C.2 DATAM errors and coding constructs

(a) DATAM errors
(b) Constructs used in the procedures

(b)

1.
```
COMMENT   RETRIEVES LINK OF INVOLVEMENT OF
          OBJECT(OBJ) IN EVENT(EV) INTO INVLNK;

PROCEDURE GETINVNAME(OBJ,EV,INVLNK);
VALUE OBJ,EV,INVLNK;
POINTER OBJ,EV,INVLNK;
BEGIN LABEL PREXIT;EBCDIC ARRAY ATSTR[0:60];

  DEFINE S(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,2); GETSNAMES(SN,2);

  SEARCH(EVNT,STSRCH);
  SEARCH(DESOB,S(0));
  SEARCH(E,S(1));
  MATCH(STSRCH,EV);
  IF NOT SUCCESS THEN ERROR(90);                          %<90>
    MATCH(S(0),OBJ);
    IF NOT SUCCESS THEN ERROR(25);                        %<25>
      REPLACE ATSTR[0] BY ATARG(EVEV,STSRCH,EVOBJ,S(0));
      SETAT(AT,ATSTR,D,E,S(1));
      GET(ASSC,EVLNK,S(1),INVOL);
      MAKARG(RESULT,INVLNK);
PREXIT;
  ENDS(S(1));ENDS(S(0));
  ENDS(STSRCH);

  RETSCH(SN,2);
END;
```

2. CURRENT(CURSOR)::= GET(D,D,CURSOR,D)

3. ATTRISUSED(VAL,ATTRI)::= MEMBER(VAL,ATTRI)

4. SETARGFILE - sets up input data files from
   information passed in command string

5. ERROR,ERRORD - error handling routines

6. Algol DEFINEs for argument construction:
```
BGNARG::= "<"
ENDARG::= ">"
DTMARG(PARG)::= "<",PARG UNTIL IN {#,>},">"
FIRSTARG(PARG)::= BGNARG,DTMARG(PARG)
LASTARG(PARG)::= DTMARG(PARG),ENDARG
ATARG(L1,C1,L2,C2)::=
     BGNARG,FIRSTARG(L1),LASTARG(C1),
     FIRSTARG(L2),LASTARG(C2),ENDARG
```

7. CALLCRENTY,CALLCRSNTY,CALLCRRELN,
   CALLDSATTR,CALLCREVNT,CALLDSRELN - are
   calls to other DATAM routines;
   e.g. CALLCRENTY invokes CRENTY.

8. ERRSTAT - PRIMDAS error status code

(a)

```
COMMENT   ENTER RANGE
          INPUT:  NAME,TYPE AND LENGTH OF RANGE;

PROCEDURE CRANGE(A); %<NAME,RTYPE,RLENGTH>
VALUE A;
POINTER A;
BEGIN LABEL PREXIT;

  POINTER NAME,RTYPE,RLENGTH;
  DEFINE CURSOR=SNAMES[SN[0]]#;
  SEARCHNAMES(SN,1);
  GETSNAMES(SN,1);
  SETARGOF(NAME,A);
  SETARGAFTER(RTYPE,NAME);
  SETARGAFTER(RLENGTH,RTYPE);

  IF MEMBER(NAME,RANGE) THEN ERROR(5)                          %<05>
  ELSE BEGIN
    SEARCH(RANGE,CURSOR);
    ENTER(RANGE,NAME,CURSOR);
    APPEND(D,TYPE,CURSOR ,RTYPE);
    APPEND(D,LNGTH,CURSOR ,RLENGTH);
    ENDS (CURSOR )
  END;

PREXIT:RETSCH(SN,1);
END;
```

(b)

```
COMMENT   LINK RANGE TO ITS SUPER-RANGE,
          INPUT:  RANGE NAME,
                  SUPER-RANGE;

PROCEDURE SETSUPR(A);%SP<NAME,SUPATR>
VALUE A;POINTER A;
BEGIN LABEL PREXIT;

  BOOLEAN MATCH1;
  POINTER NAME,SUPATR;
  DEFINE
    SNAME(I)=SNAMES[S[I]]#;
  SEARCHNAMES(S,1);GETSNAMES(S,1);
  SETARGOF(NAME,A);
  SETARGAFTER(SUPATR,NAME);

  SEARCH(RANGE,STSRCH);
  MATCH(STSRCH,NAME);
  MATCH1:=SUCCESS;
  SEARCH(RANGE,SNAME(0));
  MATCH(SNAME(0),SUPATH);
  IF NOT SUCCESS OR NOT MATCH1 THEN ERROR(10);                 %<10>
    LINK(CONTN,SNAME(0),STSRCH);
PREXIT:ENDS(SNAME(0));
  ENDS(STSRCH);

  RETSCH(S,1);
END;
```

(c)

```
COMMENT   LINK RANGE TO ITS SUB-RANGES,
          INPUT:  RANGE NAME,
                  LIST OF SUB-RANGES;
PROCEDURE SETSUBR(A); %S<NAME,SUBATLIST<SUBA>>
VALUE A;POINTER A;
BEGIN LABEL PREXIT;

  POINTER NAME,SUBATLIST,SUBA;
  DEFINE
    SNAME(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,1);GETSNAMES(SN,1);
  SETARGOF(NAME,A);
  SETARGAFTER(SUBATLIST,NAME);
  SETARGOF(SUBA,SUBATLIST);

  SEARCH(RANGE,STSRCH);
  MATCH(STSRCH,NAME);
  IF NOT SUCCESS THEN ERROR(1061);
    SEARCH(RANGE,SNAME(0));
    IF EXIST(SUBATLIST)THEN
    DO BEGIN LABEL PREXIT;

      MATCH(SNAME(0),SUBA);
      IF NOT SUCCESS THEN ERROR(10);                           %<10>
        LINK(CONTN,STSRCH,SNAME(0));
    PREXIT:END
    UNTIL NOMOREARG(SUBA);

    ENDS(SNAME(0));
PREXIT:ENDS(STSRCH);

  RETSCH(SN,1)
END;
```

FIGURE C.3 Range specification routines

(a) Enter range description
(b) Enter range - super-range association
(c) Enter range - sub-range associations

(a)

```
COMMENT   CONSTRUCT ENTITY-USES CRDESOB
          INPUT:  NAME AND RANGE OF ENTITY;

PROCEDURE CRENTY(A); %<NAME,RNGENM>
VALUE A;
POINTER A;
BEGIN
  POINTER NAME,RNGENM;

  SETARGOF(NAME,A);
  SETARGAFTER(RNGENM,NAME);
  CRDESOB(NAME,RNGENM,ENTY);
 END;
```

(b)

```
COMMENT   CONSTRUCT DESCRIBABLE OBJECT
          TOKEN INSTANCE SET
          INPUT:  NAME,RANGE AND TYPE
                  (ENTITY OR EVENT) OF OBJECT.
        USED BY CRENTY AND CREVNT;

PROCEDURE CRDESOB(NAME,RNGENM,OBJTYPE);
VALUE NAME,RNGENM,OBJTYPE;
POINTER NAME,RNGENM,OBJTYPE;
BEGIN LABEL PREXIT;

   DEFINE
    SNAME(I)=SNAMES[SN[I]]#;
  EBCDIC ARRAY RTYPE[0:10],RLENGTH[0:10];
  SEARCHNAMES(SN,1);
  GETSNAMES(SN,1);

  IF NOT EXIST(RNGENM) THEN
    IF OBJTYPE = "IEVENT" THEN BEGIN
      REPLACE RTYPE[0] BY "I#";
      REPLACE RLENGTH[0] BY "##"; END
    ELSE ERROR(65)                                          %<65>

  ELSE BEGIN
  SEARCH(RANGE,STSRCH);
  MATCH(STSRCH,RNGENM);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(65) END;                            %<65>
  GET(ASSC,0,STSRCH,TYPE);
  MAKARG(RESULT,RTYPE);
  IF RTYPE="A" THEN BEGIN
  GET(ASSC,0,STSRCH,LNGTH);
  MAKARG(RESULT,RLENGTH);
  END;
  END;

  CREATE(VSET,NAME,RTYPE,RLENGTH);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(15) END;                            %<15>

  SEARCH(DESOB,SNAME(0));
  SEARCH(VMSET,STSRCH);
  MATCH(STSRCH,NAME);
  MARK(0,0,STSRCH,0,DESOB);
  SETAT(ASSC,0,STSRCH,DESOB,SNAME(0));
  ENDS(STSRCH);
  MARK(0,0,SNAME(0),0,OBJTYPE);
  IF EXIST(RNGENM) THEN
  LINK(OBRNG,SNAME(0),STSRCH);
  ENDS(SNAME(0));

  ENDS(STSRCH);

PREXIT:RETSCH(SN,1)
END;
```

(c)

```
COMMENT    CONSTRUCT EVENT - USES CRDESOB
           INPUT:  NAME, RANGE OF TOKEN, AND
                   PAIRS OF <OBJECT,CARDINALITY OF MAPPING>
                      OF INVOLVED OBJECTS
                      OF THE EVENT;

PROCEDURE CREVNT(A); %<ENAME,ERANGE,INLIST<INOBJ<OBJ,MAPCARD>>
VALUE A; POINTER A;
BEGIN LABEL PREXIT;

  POINTER ENAME,ERANGE,INLIST,INOBJ,SCNDARG,OBJ,MAPCARD;
  DEFINE
    S(I)=SNAMES[SN[I]]#;
  REAL ARRAY CURNDX[0:5];
  EBCDIC ARRAY NEWINVLNK[0:12],ARG[0:15];
  EBCDIC VALUE ARRAY FIRSTLINK("-1##");

  INTEGER I,J;
  SEARCHNAMES(SN,8); GETSNAMES(SN,8);
  SETARGOF(ENAME,A); SETARGAFTER(ERANGE,ENAME);
  SETARGAFTER(INLIST,ERANGE);
  SETARGOF(INOBJ,INLIST); SETARGAFTER(SCNDARG,INOBJ);


  I:=-1;
  IF NOT EXIST(SCNDARG) THEN ERROR(70);                          %<70>
    DO BEGIN
      SETARGOF(OBJ,INOBJ);
      SETARGAFTER(MAPCARD,OBJ);
      I:=*+1;
      SEARCH(DESOB,S(I));
      MATCH(S(I),OBJ);
      IF NOT SUCCESS THEN BEGIN FOR J:=0
        STEP 1 UNTIL I DO ENDS(S(J));
      ERROR(25) END;                                             %<25>
    END
    UNTIL NOMOREARG(INOBJ);

    CRDESOB(ENAME,ERANGE,EVNT);
    IF NOT SUCCESS THEN BEGIN FOR J:=0 STEP 1
        UNTIL I DO ENDS(S(J)); ERROR(15);                        %<15>
        END;

      SEARCH(EVNT,S(I+1));
      MATCH(S(I+1),ENAME);
      SEARCH(E,S(I+2)); LAST(S(I+2));
      IF NOT SUCCESS THEN PUT(FIRSTLINK,CURNDX,0,1)
      ELSE
       MOVE (RESULT,CURNDX);
      SEARCH(INVOL,S(I+3));
      RESETARG(INOBJ);
      SEARCH(FLINK,D);
      FOR J:=0 STEP 1 UNTIL I
      DO BEGIN
        SETARGOF(OBJ,INOBJ);
        SETARGAFTER(MAPCARD,OBJ);
        CURNDX[3]:=*+1;
        MAKARG(CURNDX,ARG);
        ENTER(E,ARG,S(I+2));
        LINK(EVEV,S(I+1),S(I+2));
        LINK(EVOBJ,S(J),S(I+2));
        MKEINVNAME(NEWINVLNK);
        CREATE(LSET,NEWINVLNK,OBJ,ENAME);
        MATCH(D,NEWINVLNK);
          MARK(D,D,D,D,INVOL);
          SETAT(ASSC,D,D,INVOL,S(I+3));
          LINK(EVLNK,S(I+2),S(I+3));
          APPEND(D,INUM,S(I+2),MAPCARD);
        ENDS(S(J));
        NEXTARG(INOBJ);
      END;
      ENDS(D); ENDS(S(I+3));
      ENDS(S(I+2)); ENDS(S(I+1));

PREXIT: RETSCH(SN,8)
END;

%
```

FIGURE C.4 Describable object construction

(a) Entity construction main program
(b) Construction routine
(c) Event construction

```
COMMENT   CONSTRUCT SUBOBJECT
          INPUT:  NAME OF SUBOBJECT,
                  SOURCE OBJECT;

PROCEDURE CRSNTY(A); %<NAME,OFOBJ>
VALUE A;
POINTER A;
BEGIN LABEL PREXIT;

  POINTER NAME,OFOBJ;
  DEFINE
    SNAME(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,2);
  GETSNAMES (SN,2);
  SETARGOF(NAME,A);
  SETARGAFTER(OFOBJ,NAME);

  SEARCH(DESOB,STSRCH);
  MATCH(STSRCH,OFOBJ);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(20) END;                                  %<20>
  CREATE(MSET,NAME,OFOBJ,D);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(15) END;                                  %<15>
  SEARCH(SUBOB,SNAME(0));

    SEARCH(DESOB,SNAME(1));
    SEARCH(VMSET,STSRCH);
    MATCH(STSRCH,NAME);
    MARK(D,D,STSRCH,D,DESOB);
    SETAT(ASSC,D,STSRCH,DESOB,SNAME(1));
    MARK(D,D,SNAME(1),U,SUBOB);
    SETAT(ASSC,D,SNAME(1),SUBOB,SNAME(0));
    ENDS(STSRCH);
    ENDS (SNAME(1));

  LINK(HASUB,STSRCH,SNAME(0));
  ENDS(SNAME(0));
  ENDS(STSRCH);

PREXIT:RETSCH(SN,2)
END;
```

FIGURE C.5 Sub-object construction

(a)

```
COMMENT   CONSTRUCT ATTRIBUTE
          INPUT:  DESCRIBED OBJECT,
                  NAME, RANGE, AND
                  CARDINALITY OF MAPPING
                        OF THE ATTRIBUTE AND ATTRIBUTE ASSOCIATION.
          USES ATRLNK;

PROCEDURE CRATTR(A);%<OFOBJ,NAME,RANGE,CARDMAP>
VALUE A; POINTER A;
BEGIN POINTER OFOBJ,NAME,RANGE,CARDMAP;

  SETARGOF(OFOBJ,A);
  SETARGAFTER(NAME,OFOBJ);
  SETARGAFTER(RANGE,NAME);
  SETARGAFTER(CARDMAP,RANGE);
  ATRLNK(OFOBJ,NAME,RANGE,CARDMAP)
END;

COMMENT   CONSTRUCT COMPOSITE ATTRIBUTE OF EXISTING ATTRIBUTES
          INPUT:  DESCRIBED OBJECT,
                  NAME,
                  CARDINALITY OF MAPPING, AND
                  LIST OF COMPONENTS
                        OF THE ATTRIBUTE
          USES ATRLNK;

PROCEDURE SETCMPO(A);%<OFOBJ,NAME,CARDMAP,MBRLIST<MBR>>
VALUE A; POINTER A;
BEGIN LABEL PREXIT;

  POINTER OFOBJ,NAME,CARDMAP,MBRLIST,MBR;
  DEFINE
    SNAME(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,1);
  GETSNAMES(SN,1);
  SETARGOF(OFOBJ,A);
  SETARGAFTER(NAME,OFOBJ);
  SETARGAFTER(CARDMAP,NAME);
  SETARGAFTER(MBRLIST,CARDMAP);
  SETARGOF(MBR,MBRLIST);


  ATRLNK(OFOBJ,NAME,D,CARDMAP);
  IF NOT SUCCESS THEN ERROR(1041);
  SEARCH(ATTRI,STSRCH);
  MATCH(STSRCH,NAME);
  SEARCH(ATTRI,SNAME(0));
  IF EXIST(MBRLIST)THEN
    DO BEGIN LABEL PREXIT;

      MATCH(SNAME(0),MBR);
      IF NOT SUCCESS THEN ERROR(35);                              %<35>
      LINK(CMPOS,STSRCH,SNAME(0));
    PREXIT:
    END
    UNTIL NOMOREARG(MBR);

  ENDS(SNAME(0));
  ENDS(STSRCH);

PREXIT:RETSCH(SN,1)
END;
```

(b)

```
COMMENT   ATTRIBUTE CONSTRUCTION FOR
          CRATTR AND SETCHPO;

PROCEDURE ATRLNK(OFOBJ,NAME,RNGENM,CARDMAP);
VALUE OFOBJ,NAME,RNGENM,CARDMAP;
POINTER OFOBJ,NAME,RNGENM,CARDMAP;
BEGIN DEFINE SNAME(I)=SNAMES[SN[I]]#;

  LABEL PREXIT;
  EBCDIC ARRAY RTYPE[0:10],RLENGTH[0:10];
  SEARCHNAMES(SN,3);
  GETSNAMES(SN,3);

  SEARCH(DESOB,SNAME(1));
  MATCH(SNAME(1),OFOBJ);
  IF NOT SUCCESS THEN BEGIN
    ENDS(SNAME(1));ERROR(25)END;                          %<25>

  IF EXIST(RNGENM)THEN
  BEGIN
    SEARCH(RANGE,STSRCH);
    MATCH(STSRCH,RNGENM);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
      ENDS(SNAME(1));ERROR(10)END;                        %<10>
    GET(ASSC,D,STSRCH,TYPE);
    MAKARG(RESULT,RTYPE);
    IF RTYPE = "A" THEN BEGIN
    GET(ASSC,D,STSRCH,LNGTH);
    MAKARG(RESULT,RLENGTH);
    END;
    CREATE(VSET,NAME,RTYPE,RLENGTH);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
      ENDS(SNAME(1));ERROR(15)END;                        %<15>
    DISASS(OFOBJ,NAME);
  END;

  SEARCH(HASAT,SNAME(2));
  SEARCH(ATTRI,SNAME(0));
  ENTER(ATTRI,NAME,SNAME(0));
  COMLNK(HASAT,SNAME(2),AOBJ ,SNAME(1),AA,SNAME(0));
  APPEND(D,ANUM,SNAME(2),CARDMAP);
  ENDS(SNAME(2));
  IF EXIST(RNGENM)THEN
  BEGIN
    LINK(ATRNG,SNAME(0),STSRCH);
    ENDS(SNAME(0));
    ENDS(STSRCH);
  END;
  ENDS(SNAME(1));

PREXIT:RETSCH(SN,3)
END;
```

FIGURE C.6 Attribute construction
(a) Main routines - single and composite attributes
(b) Construction routine

```
COMMENT   CONSTRUCT RELATIONSHIP
          INPUT:  NAME,
                  THE TWO OBJECTS, AND
                  CARDINALITY OF MAPPING
                     OF THE RELATIONSHIP;

PROCEDURE CRRELN(A); %<RNAME,OBJ1,OBJ2,CARDMAP>
VALUE A; POINTER A;
BEGIN LABEL PREXIT;

  POINTER RNAME,OBJ1,OBJ2,CARDMAP;
  DEFINE S(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,3); GETSNAMES(SN,3);
  SETARGOF(RNAME,A); SETARGAFTER(OBJ1,RNAME);
  SETARGAFTER(OBJ2,OBJ1); SETARGAFTER(CARDMAP,OBJ2);

  SEARCH(DESOB,D);
  MATCH(D,OBJ1);
  IF NOT SUCCESS THEN BEGIN ENDS(D); ERROR(25) END;       %<25>
    SEARCH(DESOB,S(0));
    MATCH(S(0),OBJ2);
    IF NOT SUCCESS THEN BEGIN ENDS(D); ENDS(S(0));
       ERROR(25) END;                                     %<25>
      CREATE(LSET,RNAME,OBJ1,OBJ2);
      IF NOT SUCCESS THEN BEGIN ENDS(D);
        ENDS(S(0)); ERROR(15) END;                        %<15>
      SEARCH(R,S(1));
      COMLNK(R,S(1),RTO,D,RWITH,S(0));
    ENDS(S(0));
  ENDS(D);

  SEARCH(RELN,S(0));
    SEARCH(FLINK,D);
    MATCH(D,RNAME);
    MARK(D,D,D,D,RELN);
    SETAT(ASSC,D,D,RELN,S(0));
    ENDS(D);
      SEARCH(HASR,S(2));
      COMLNK(HASR,S(2),ROBJ,S(1),RR,S(0));
      APPEND(D,RNUM,S(2),CARDMAP);
      ENDS(S(2));
  ENDS(S(0));

      ENDS(S(1));
      ENDS(S(1));

PREXIT:RETSCH(SN,3)
END;
```

FIGURE C.7 Relationship construction

```
COMMENT   ENTER TOKENS - THREE MODES;
            (1)  FILEINPUT= FILE OF TOKENS
            (2)  SINGLEINPUT= TOKEN IN COMMAND ARRAY
            (3)  NOCHECK= OBJECT-ATTRIBUTE INSTANCES
          INPUT;  CBUF= OBJECT NAME AND
                        APPROPRIATE PARAMETERS FOR INPUT MODE
                  SOURCE= INPUT MODE;

PROCEDURE NTROBJ(CBUF,SOURCE); %<<ENTOBJ,TFILE/TOKEN>
VALUE CBUF,SOURCE;
POINTER CBUF; INTEGER SOURCE;
BEGIN INTEGER RECSIZE;

  LABEL PREXIT;
  FILE INFILE(KIND=READER);

  POINTER ENTOBJ,TFILE;
  DEFINE
    S(I)=SNAMES[SN[I]]#,
    FILEINPUT=0#,SINGLEINPUT=1#,
    NOCHECK=2#,
    TOKEN=TFILE#;
  SEARCHNAMES(SN,1); GETSNAMES(SN,1);
  SETARGOF(ENTOBJ,CBUF);
  SETARGAFTER(TFILE,ENTOBJ);

  IF NOT MEMBER(ENTOBJ,DESOB) THEN ERROR(25);                    %<25>
    SEARCH(ENTOBJ,S(0));

    CASE SOURCE OF
    BEGIN

    FILEINPUT;          % MULTIPLE ENTER
      SETARGFILE(INFILE,TFILE);
      IF INFILE.PRESENT THEN
      BEGIN
      RECSIZE:=INFILE.MAXRECSIZE*6;
      BEGIN
        EBCDIC ARRAY BUF[0:RECSIZE+2];

        POINTER TOKEN;
        REPLACE BUF[RECSIZE] BY "##";
        WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
        DO BEGIN
          LABEL PREXIT;

          SETONARG(TOKEN,BUF);

          MATCH(S(0)  ,TOKEN);
        IF SUCCESS THEN ERROR0 (40);                             %<40>
          ENTER(ENTOBJ,TOKEN,0);

        PREXIT;END
      END

    END;

    SINGLEINPUT;          % COMMAND ARRAY INPUT

      MATCH(S(0)  ,TOKEN);
      IF SUCCESS THEN ERROR(40);                                 %<40>
      ENTER(ENTOBJ,TOKEN,0);

    NOCHECK;          % <<APPFIL><<FILEKIND><FILETITLE>>
      BEGIN
      POINTER APPFIL,FILEDESC;

      INTEGER RECSIZE;
      SETARGOF(APPFIL,TFILE);
      SETARGAFTER(FILEDESC,APPFIL);
      SETARGFILE(INFILE,FILEDESC);
      IF INFILE.PRESENT THEN
      BEGIN
      RECSIZE:=INFILE.MAXRECSIZE*6;
      BEGIN % <<TOKEN><APPARGS>>
        EBCDIC ARRAY BUF[0:RECSIZE+2];

        POINTER TOKEN,APPARGS;
        REPLACE BUF[RECSIZE] BY "##";
        WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
        DO BEGIN
          SETONARG(TOKEN,BUF);
          SETARGOF(TOKEN,TOKEN);
          SETARGAFTER(APPARGS,TOKEN);

          MATCH(S(0),TOKEN);
          IF NOT SUCCESS THEN ENTER(ENTOBJ,TOKEN,S(0));
          APPEND(ENTOBJ,APPFIL,S(0),APPARGS);

        END
      END END

    END ELSE;END;

    ENDS(S(0));
PREXIT; RETSCH(SN,1);
END;
```

FIGURE C.8 Entry of tokens

```
COMMENT   ENTER ATTRIBUTE INSTANCES - TWO MODES:
            (1)  FILEINPUT= FILE OF <VALUE,TOKEN> PAIRS
            (2)  SINGLEINPUT= ONE PAIR OF <VALUE,TOKEN>
                                IN COMMAND ARRAY
          INPUT:  CBUF= ATTRIBUTE TYPE,
                        DESCRIBED OBJECT AND
                        APPROPRIATE PARAMETERS FOR INPUT MODE
                  SOURCE= INPUT MODE;

PROCEDURE NTRATR(CBUF,SOURCE);%<<ATTRIBUTE,OFOBJECT,<VAL,TOKEN>>
VALUE CBUF,SOURCE;
POINTER CBUF; INTEGER SOURCE;
BEGIN
  SEARCHNAMES(SN,1);
%
  PROCEDURE APPENDATR(VAL,TOKEN);
  VALUE VAL,TOKEN; POINTER VAL,TOKEN;
  BEGIN LABEL PREXIT; BOOLEAN STILLOK;


    STILLOK:=TRUE;
    IF MAPNP EQL "1:1" THEN
      IF OBJHASATTR  (OFOBJECT,TOKEN,ATTRIBUTE)
        OR ATTRISUSED(VAL,ATTRIBUTE)
        THEN STILLOK:=FALSE ELSE
    ELSE
    IF MAPNP EQL "N:1" THEN
      IF OBJHASATTR(OFOBJECT,TOKEN,ATTRIBUTE)
        THEN STILLOK:=FALSE ELSE
    ELSE
    IF MAPNP EQL "1:N" THEN
      IF ATTRISUSED(VAL,ATTRIBUTE)
        THEN STILLOK:=FALSE;

    IF STILLOK THEN BEGIN
      MATCH(STSRCH,TOKEN);
      IF NOT SUCCESS THEN ERROR(55);                          %<55>
      APPEND(OFOBJECT,ATTRIBUTE,STSRCH,VAL);
    END
    ELSE ERROR(45);                                           %<45>

  PREXIT: END;


  GETSNAMES(SN,1);

  SEARCH(DESOB,STSRCH);
  MATCH(STSRCH,OFOBJECT);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
     ERROR(25)END;                                            %<25>
    SEARCH(ATTRI,SNAME(0));
    MATCH(SNAME(0),ATTRIBUTE);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
       ENDS(SNAME(0)); ERROR (35) END;                        %<35>
      REPLACE FORATARG[0] BY
        ATAHG(STSRCH,AA,SNAME(0),AOBJ);
      GET(AT,FORATARG,D,ANUM);
      MAKARG(RESULT,MAPNP);
      ENDS(STSRCH); ENDS(SNAME(0));
      SEARCH(OFOBJECT,STSRCH);

      CASE SOURCE OF
      BEGIN

      FILEINPUT:
          WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
          DO BEGIN
            LABEL PREXIT;

            APPENDATR(VAL,TOKEN)
          END

      SINGLEINPUT:
        APPENDATR(VAL,TOKEN)       % COMMAND ARRAY INPUT

      ELSE:; END;

      ENDS(STSRCH);

PREXIT: RETSCH(SN,1)
END;
```

FIGURE C.9 Enter attribute instances

```
COMMENT   ENTER SUBOBJECT-ATTRIBUTE INSTANCES
          NO CARDINALITY CHECK.
          INPUT:  CBUF= SUBOBJECT NAME,
                        LIST OF ATTRIBUTES,
                        INPUT FILE INFORMATION.
                  SOURCE= ONLY NOCHECK MODE;

PROCEDURE NTRSUB(CBUF,SOURCE);%<<SOBJ><ARGS>>
VALUE CBUF,SOURCE;
POINTER CBUF;INTEGER SOURCE;
BEGIN LABEL PREXIT;


  SEARCH(SUBOB,STSRCH);
  MATCH(STSRCH,SOBJ);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
     ERROR(60)END;                                            %<60>
    ASSOC(HASUB,STSRCH,DESOB);
    MAKARG(RESULT,OFOBJ);
  ENDS(STSRCH);
  SEARCH(OFOBJ,STSRCH);
  SEARCH(SOBJ,S(0));

    CASE SOURCE OF
    BEGIN

  NOCHECK:          %<<APPFIL><<FILEKIND><FILETITLE>>>
        WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
        DO BEGIN  %<<TOKEN><APPARGS>>
          LABEL PREXIT;


          MATCH(STSRCH,TOKEN);
          IF NOT SUCCESS THEN ERROR(55);                      %<55>
            MARK(D,D,STSRCH,D,SOBJ);
            SETAT(ASSC,D,STSRCH,SUBJ,S(0));
            APPEND(SOBJ,APPFIL,S(0),APPARGS);

          PREXIT: END
        END

    ELSE:;END;

  ENDS(STSRCH);ENDS(S(0));

PREXIT: RETSCH(SN,1);
END;
```

FIGURE C.10 Enter sub-object - attribute instances

(a)

```
COMMENT   ENTER RELATIONSHIP
          NO CARDINALITY CHECK
          INPUT:  CBUF= RELATIONSHIP NAME,
                        THE TWO RELATED OBJECTS AND
                        INPUT FILE INFORMATION
                  SOURCE= ONLY NOCHECK MODE;

PROCEDURE NTRELN(CBUF,SOURCE); %<RNAME,OBJ1,OBJ2,ARGS>
VALUE CBUF,SOURCE;
POINTER CBUF; INTEGER SOURCE;
BEGIN

    SEARCH(RELN,STSRCH);
    MATCH(STSRCH,RNAME);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
       ERROR(30) END;                                              %<30>
      REPLACE LINKARG[0] BY FIRSTARG(RR),OTHARG(ROBJ),
        LASTARG(RTO);
      ASSOC(LINKARG,STSRCH,DESOB);
      MOVE(RESULT,TEMP);
      REPLACE LINKARG[0] BY FIRSTARG(RR),OTHARG(ROBJ),
        LASTARG(RWITH);
      ASSOC(LINKARG,STSRCH,DESOB);
      PUTSAME(RESULT,OBJ1,ROBJ1);PUTSAME(RESULT,OBJ2,ROBJ2);


      IF NOT((EQ(ROBJ1,TEMP) AND EQ(ROBJ2,RESULT)) OR
        (EQ(ROBJ2,TEMP) AND EQ (ROBJ1,RESULT))) THEN
        BEGIN ENDS(STSRCH); ERROR(50) END;                         %<50>
        SEARCH(OBJ1,S(0));
        SEARCH(OBJ2,S(1));

        CASE SOURCE OF
        BEGIN

        NOCHECK:          %<<FILEKIND><FILETITLE>>
          BEGIN %<<TOK1><TOK2>>
            WHILE NOT READ(INFILE,<C* >,RECSIZE,BUF[0])
            DO BEGIN LABEL PREXIT;


              MATCH(S(0),TOK1);
              IF NOT SUCCESS THEN ERRORD(55);                      %<55>
                MATCH(S(1),TOK2);
                IF NOT SUCCESS THEN ERRORD(55);                    %<55>
                  LINK(RNAME,S(0),S(1));

            PREXIT: END
          END

        ELSE:; END;

        ENDS(S(1)); ENDS(S(0));
    ENDS(STSRCH);

PREXIT: RETSCH(SN,2)
END;
```

(b)

```
COMMENT  ENTER RELATIONSHIP INVOLVING EVENT
           IN TERMS OF ITS INVOLVED OBJECTS.
         INPUT:  RELATIONSHIP NAME,
                 THE TWO EVENT MEMBERS,
                 THE OTHER OBJECT INVOLVED,
                 INPUT FILE INFORMATION;

PROCEDURE RELNWITHEVNT(CBUF); %(RN,EN(011,012),02,FILEDESC)
VALUE CBUF;POINTER CBUF;
BEGIN
  EBCDIC ARRAY INV1[0:15],INV2[0:15],ATSTR[0:75];

  INTEGER I;
  POINTER RNAME,EOBJ,OBJ2,FILEDESC,ENAME,OBJS,
    OBJ11,OBJ12;
  DEFINE
    S(I)=SNAMES[SN[I]]#;
  SEARCHNAMES(SN,4);GETSNAMES(SN,4);

  SEARCH(OBJ11,S(0));
  SEARCH(OBJ12,S(1));
  SEARCH(OBJ2,S(2));
  SEARCH(ENAME,S(3));
  GETINVNAME(OBJ11,ENAME,INV1);
  GETINVNAME(OBJ12,ENAME,INV2);
  REPLACE ATSTR[0] BY ATARG(INV1,S(0),INV2,S(1));
    BEGIN %<<<OBJ11><OBJ12>><OBJ2>>
      WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
      DO BEGIN LABEL PREXIT;


        MATCH(S(0),OBJ11);
         IF NOT SUCCESS THEN ERROR(55);                            %<55>
          MATCH(S(1),OBJ12);
            IF NOT SUCCESS THEN ERROR(55);                         %<55>
              SETAT(AT,ATSTR,0,ENAME,S(3));
               IF NOT SUCCESS THEN ERROR(85);                      %<85>
                 MATCH(S(2),OBJ2);
                   IF NOT SUCCESS THEN ERROR(55);                  %<55>
                     LINK(RNAME,S(3),S(2));
      PREXIT:END
    END

  FOR I:=0 STEP 1 UNTIL 3
  DO ENDS(S(I));

  RETSCH(SN,4);
END;
```

FIGURE C.11 Enter relationships

(a) In terms of the directly involved objects

(b) Involving an event and in terms of the members of the event

```
COMMENT    ENTER EVENT-ATTRIBUTE INSTANCES
           NO CARDINALITY CHECK
           INPUT:  CBUF= EVENT NAME,
                         LIST OF INVOLVED OBJECTS,
                         LIST OF ATTRIBUTES,
                         INPUT FILE INFORMATION
                   SOURCE= ONLY NOCHECK MODE;

PROCEDURE NTREVT(CBUF,SOURCE);%<<ENAME><INVOBJS><ARGS>>
VALUE CBUF,SOURCE;
POINTER CBUF; INTEGER SOURCE;
BEGIN POINTER ENAME,INVOBJS,ARGS,OBJ,PATCS;

  SEARCH(EVNT,STSRCH);
  MATCH(STSRCH,ENAME);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
     ERROR(90) END;                                                    %<90>
    SETARGOF(OBJ,INVOBJS);
    I:=2;
    SEARCH(DESOB,S(0));
    SEARCH(E,S(1));
    SEARCH(ENAME,S(2));
    REPLACE PATCS:ATCS[0] BY BGNARG;
    DO BEGIN MATCH(S(0),OBJ);
      IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
          FOR J:=0 STEP 1 UNTIL I DO ENDS(S(J));
          ERROR(25) END;                                               %<25>
        I:=*+1;
        SEARCH(OBJ,S(I));
        REPLACE LARG[0] BY ATARG(EVOBJ,S(0),EVEV,STSRCH);
        SETAT(AT,LARG,D,E,S(1));
        ASSOC(EVLNK,S(1),INVOL);
        MAKARG(RESULT,INVLNK[I,0]);
        REPLACE PATCS:PATCS     BY FIRSTARG(INVLNK[I,0]),
          LASTARG(S(I));
    END
    UNTIL NOMOREARG(OBJ);
    REPLACE PATCS     BY ENDARG;


      CASE SOURCE OF
      BEGIN

      NOCHECK:          %<ATTRIBUTE VMSETS><FILEDESC>
        BEGIN  %<<TOK1>,...,<TOKN>><<ATTR1>,...,<ATTRN>>
          WHILE NOT READ(INFILE,<C*>,RECSIZE,BUF[0])
          DO BEGIN LABEL PREXIT;

            IF NOT EXIST(TOKEN) THEN ERRORD(95);                       %<95>
              FOR J:=3 STEP 1 UNTIL I DO BEGIN

                MATCH(S(J),TOKEN);
                IF NOT SUCCESS THEN ERRORD(55);                        %<55>
                  NEXTARG(TOKEN);
                  IF NOT SUCCESS AND J < I THEN ERRORD (95);           %<95>
              END;
              SETAT(AT,ATCS,D,ENAME,S(2));
              IF NOT SUCCESS THEN
              BEGIN EBCDIC ARRAY NEWTOKEN[0:10];

                LAST(S(2));
                IF NOT SUCCESS THEN
                  REPLACE NEWTOKEN BY "0#*"
                ELSE BEGIN
                  RESULT[3]:=*+1;
                  MAKARG(RESULT,NEWTOKEN);
                END;
                ENTER(ENAME,NEWTOKEN,S(2));
                FOR J:=3 STEP 1 UNTIL I DO
                  LINK(INVLNK[J,0],S(J),S(2));
              END;

              APPENDM(ENAME,ATTVMSETS,S(2),ATTR );

          PREXIT:END  %<READ>
      END

      ELSE:;

      END; %<CASE>
      FOR J:=0 STEP 1 UNTIL I
      DO ENDS(S(J));
      ENDS(STSRCH);

PREXIT:RETSCH(SN,8);
END;
```

FIGURE C.12 Enter event-attribute instances

```
COMMENT   DESTROY DESCRIBABLE OBJECT
          SUBROUTINES:  DSOA,
                        DSOR,
                        DSISA,
                        RDSOBJ,
          INPUT:  NAME OF OBJECT;

PROCEDURE DSOBJ(NAME);%<NAME>
VALUE NAME; POINTER NAME;
BEGIN EBCDIC ARRAY ARG[0:20];

  DEFINE DESOBCURSOR=SNAMES[S[0]]#;
  SEARCHNAMES(S,1);

  PROCEDURE RDSOBJ(OBJ);
  VALUE OBJ; POINTER OBJ; FORWARD;
  PROCEDURE DSOA;%DESTROY ATTRIBUTES OF OBJ
  BEGIN DEFINE M(I)=FNAMES[F[I]]#;

  EBCDIC VALUE ARRAY TOATTR("<<1AOBJ><1AA>>");

  EBCDIC ARRAY ATTRIBUTE[0:20];
    FILENAMES(F,1);GETFNAMES(F,1);

    CREATE(MSET,M(0),ATTRI,D);
    GETLNK(TOATTR,DESOBCURSOR,ATTRI,M(0));

    SEARCH(M(0),STSRCH);
    FIRST(STSRCH);
    WHILE NOT EOF(STSRCH)DO BEGIN
      MAKARG(RESULT,ARG);
      REPLACE ATTRIBUTE BY DTMARG(ARG);
      NEXT(STSRCH); DSATTR(ATTRIBUTE)
    END;
    ENDS(STSRCH);

    DESTROY(M(0));

    RETFIL(F,1);
  END;

  PROCEDURE DSOR;%DESTROY RELNS OF OBJ
  BEGIN DEFINE M(I)=FNAMES[F[I]]#;

    FILENAMES(F,2);GETFNAMES(F,2);

    CREATE(MSET,M(0),R,D);
    CREATE (MSET,M(1),MASR,D);
    GETLNK(RTU,DESOBCURSOR,R,M(0));
    GETLNK(RWITH,DESOBCURSOR,R,M(0));

    SEARCH(M(0),STSRCH);
    FIRST(STSRCH);
    WHILE NOT EOF(STSRCH) DO BEGIN
      EMPTY(M(1));
      GETLNK(ROBJ,STSRCH,MASR,M(1));
      SEARCH(M(1),STSRCH);
      FIRST(STSRCH);
      WHILE NOT EOF(STSRCH) DO BEGIN
        DELETE(ASSC,D,STSRCH,RNUM);


        DELETE(D,D,STSRCH,D)
      END;
      ENDS(STSRCH);
      DELETE(D,D,STSRCH,D)
    END;
    ENDS(STSRCH);

    DESTROY(M(0)); DESTROY(M(1));

    RETFIL(F,2)
  END;
%
```

FIGURE C.13 Destroy describable object

```
  PROCEDURE DSISA; % DESTROY ANY SUBOBJECTS
  BEGIN DEFINE

    SETOFSUBS=FNAMES[M[0]]#;
    EBCDIC ARRAY SOBJ[0:20];
    FILENAMES(M,1);
    GETFNAMES(M,1);

    CREATE(MSET,SETOFSUBS,SUBOB,D);
    GETLNK(HASUB,DESOBCURSOR,SUBOB,SETOFSUBS);

    SEARCH(SETOFSUBS,STSRCH);
    FIRST(STSRCH);
    WHILE NOT EOF(STSRCH)DO BEGIN
      MAKARG(RESULT,SOBJ);
      RDSOBJ(SOBJ);
      NEXT(STSRCH);
    END;
    ENDS(STSRCH);

    DESTROY(SETOFSUBS);

    RETFIL(M,1)
  END;

  PROCEDURE RDSOBJ(OBJ); % RECURSIVE ROUTINE OF DSOBJ
  VALUE OBJ; POINTER OBJ;
  BEGIN LABEL PREXIT;

    DEFINE OBJEVENTS=FNAMES[FG[0]]#;
    DEFINE EVNTCURSOR=SNAMES[SG[0]]#;
    EBCDIC VALUE ARRAY TOINVOL("<<EVEV><EVLNK>>");

    SEARCHNAMES(SG,1);
    FILENAMES(FG,1);
    GETSNAMES(SG,1);
    GETFNAMES(FG,1);

    MATCH(DESOBCURSOR,OBJ);
    IF NOT SUCCESS THEN ERROR(25);                    %<25>
    DSOA;
    DSOR;
    DSISA;
    CREATE(MSET,OBJEVENTS,E,D);
    GETLNK(EVOBJ,DESOBCURSOR,E,OBJEVENTS);

    SEARCH(OBJEVENTS,STSRCH);
    FIRST(STSRCH);
    WHILE NOT EOF(STSRCH) DO BEGIN
      SEARCH(EVNT,EVNTCURSOR);
      SETAT(ASSC,EVEV,STSRCH,EVNT,EVNTCURSOR);
      NEXT(STSRCH);
      DO BEGIN
        DELETE(ASSC,EVEV,EVNTCURSOR,INUM);
        DELETE(ASSC,EVEV,EVNTCURSUR,E);
      END
      UNTIL NOT SUCCESS;
      CURRENT(EVNTCURSOR);
      MAKARG(RESULT,ARG);
      ENDS(EVNTCURSOR);
      RDSOBJ(ARG);
    END;
    ENDS(STSRCH);

    NEXT(DESOBCURSOR);
    DESTROY(OBJEVENTS);
    DESTROY(OBJ);

  PREXIT: RETFIL(FG,1);
  RETSCH(SG,1);
  END;



% START MAIN PROGRAM

  GETSNAMES(S,1);
  SEARCH(DESOB,DESOBCURSOR);
  RDSOBJ(NAME);
  ENDS(DESOBCURSOR);
END;
```

```
COMMENT   DESTROY RELATIONSHIP
          INPUT:  NAME OF RELATIONSHIP;

PROCEDURE DSRELN(RNAME);
VALUE RNAME; POINTER RNAME;
BEGIN
  LABEL PREXIT;

  SEARCH(RELN,STSRCH);
  MATCH(STSRCH,RNAME);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(30) END;
    DELETE(ASSC,RR,STSRCH,RNUM);
    DELETE(ASSC,RR,STSRCH,HASR);
    ENDS(STSRCH);
    DESTROY(RNAME);

PREXIT:
END;
```

FIGURE C.14 Destroy relationship

```
COMMENT   DESTROY ATTRIBUTE
          INPUT:  NAME OF ATTRIBUTE;

PROCEDURE DSATTR(NAME); % <NAME>
VALUE NAME; POINTER NAME;
BEGIN LABEL PREXIT;

  SEARCH(ATTRI,STSRCH);
  MATCH(STSRCH,NAME);
  IF NOT SUCCESS THEN BEGIN
    ENDS(STSRCH); ERROR(25) END;
  DELETE(ASSC,AA,STSRCH,ANUM);
  DELETE(ASSC,AA,STSRCH,HASAT);
  GET(ASSC,CMPOS,STSRCH,ATTRI);
  IF NOT SUCCESS THEN DESTROY(NAME);
  DELETE(D,D,STSRCH,D);
  ENDS(STSRCH);

PREXIT:END;
```

FIGURE C.15 Destroy attribute

(a)

```
COMMENT   DELETE TOKEN,
          SUBROUTINES:  DLOA,RDLOBJ,DLSUBOB,
          USED BY DLENTY,DLEVNT;

PROCEDURE DLOBJ(TOKEN,OBJ);
VALUE TOKEN,OBJ;POINTER TOKEN,OBJ;
BEGIN INTEGER LS,LM;

  EBCDIC ARRAY DARG[0:15],DARG2[0:15];
  REAL ARRAY TEMP[0:30];
  DEFINE DESOBCURSOR=SNAMES[SN[0]]#,
    NOTASUB=FALSE#;
  SEARCHNAMES(SN,1);

  PROCEDURE DLOA; % ATTRIBUTE DELETE
  BEGIN DEFINE SETOFATTRI=FNAMES[F[0]]#,
    SX(I)=SNAMES[S[I]]#;
    SEARCHNAMES(S,1);FILENAMES(F,1);
    EBCDIC VALUE ARRAY LARG("<<1AUBJ><1AA>>");

    GETSNAMES(S,1);GETFNAMES(F,1);

    CREATE(MSET,SETOFATTRI,ATTRI,D);
    GETLNK(LARG,DESOBCURSOR,ATTRI,SETOFATTRI);

    SEARCH(SETOFATTRI,SX(0));
    FIRST(SX(0));
    WHILE NOT EOF(SX(0))DO
    BEGIN
      GET(D,D,SX(0),D);
      MAKARG(RESULT,DARG);
      MATCH(STSRCH,TOKEN);
      WHILE SUCCESS DO
      BEGIN
        DELETE(ASSC,D,STSRCH,DARG);
        NEXTM(STSRCH)
      END;
      NEXT(SX(0))
    END;
    ENDS(SX(0));

    DESTROY(SETOFATTRI);

    RETFIL(F,1);
    RETSCH(S,1);
  END;

  PROCEDURE RDLOBJ(TOKEN,OBJ,ASUBOBJ);
  VALUE TOKEN,OBJ,ASUBOBJ; BOOLEAN ASUBOBJ;
  POINTER TOKEN,OBJ; FORWARD;

  PROCEDURE DLSUBOB; % DESTROY ANY SUBOBJECTS
  BEGIN DEFINE

    ASUBOBJ=TRUE#,
    SETOFSUBS=FNAMES[M[0]]#;
    EBCDIC ARRAY SUBJ[0:20];
    FILENAMES(M,1);
    GETFNAMES(M,1);

    CREATE(MSET,SETOFSUBS,SUBOB,D);
    GETLNK(HASUB,DESOBCURSOR,SUBOB,SETOFSUBS);
    SEARCH(SETOFSUBS,STSRCH);
    FIRST(STSRCH);
    WHILE NOT EOF(STSRCH)DO BEGIN
      MAKARG(RESULT,SOBJ);
      RDLOBJ(TOKEN,SOBJ,ASUBOBJ);
      NEXT(STSRCH);
    END;
    ENDS(STSRCH);
    DESTROY(SETOFSUBS);

    RETFIL(M,1)
  END;
```

```
PROCEDURE RDLOBJ(TOKEN,OBJ,ASUBOBJ);%RECURSIVE DELETE
VALUE TOKEN,OBJ,ASUBOBJ;
POINTER TOKEN,OBJ; BOOLEAN ASUBOBJ;
BEGIN LABEL PREXIT;

  DEFINE TEV=DARG#,TLNK=DARG2#,
    ESUBSET=FNAMES[F[0]]#,
    EVNTSUBSET=FNAMES[F[1]]#,
    ESUBCURSOR=SNAMES[S[0]]#;
  SEARCHNAMES(S,1); FILENAMES(F,2);
  GETSNAMES(S,1); GETFNAMES(F,2);

  MATCH(DESUBCURSOR,OBJ);
  IF NOT SUCCESS THEN ERROR(25);                                    %<25>
    SEARCH(OBJ,STSRCH);
    MATCH(STSRCH,TOKEN);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
       ERROR(55)END;                                                %<55>

    %DELETE EVENTS
      CREATE(MSET,ESUBSET,E,0);
      GETLNK(EVOBJ,DESUBCURSOR,E,ESUBSET);
      SEARCH(ESUBSET,ESUBCURSOR);
      FIRST(ESUBCURSOR);
      WHILE NOT EOF(ESUBCURSOR)DO BEGIN
        GET(ASSC,EVEV,ESUBCURSOR,EVNT );
        MAKARG(RESULT,TEV);
        GET(ASSC,EVLNK,ESUBCURSOR,INVOL);
        MAKARG(RESULT,TLNK);
        CREATE(MSET,EVNTSUBSET,TEV,0);
        GETLNK(TLNK,STSRCH,TEV,EVNTSUBSET);
        SEARCH(EVNTSUBSET,STSRCH);
        FIRST(STSRCH);
        WHILE NOT EOF(STSRCH) DO BEGIN
          GET(D,D,STSRCH,D);
          MAKARG(RESULT,DARG2);
          NEXT(STSRCH);
          RDLOBJ(DARG2,TEV,NOTASUB);

        END;
        ENDS(STSRCH);
        DESTROY(EVNTSUBSET);
        NEXT(ESUBCURSOR)
      END;
      ENDS(ESUBCURSOR);
      DESTROY(ESUBSET);
    %
      DLSUBOB;
      DLOA;
    %
    % DELETE OBJECT
      IF NOT ASUBOBJ THEN
      BEGIN
      MATCH(STSRCH,TOKEN);
      CURRENT(STSRCH);
      MOVE(RESULT,TEMP);
    % WHILE EQ(RESULT,TEMP)DO
      BEGIN
        DELETE(D,D,STSRCH,D);
        GET(D,D,STSRCH,D)
      END;
      END;
      ENDS(STSRCH);
  PREXIT: RETSCH(S,1); RETFIL(F,2);
  END;

%
  GETSNAMES(SN,1);

  SEARCH(DESOB,DESOBCURSOR);
  RDLOBJ(TOKEN,OBJ,NOTASUB);
  ENDS(DESOBCURSOR);

  RETSCH(SN,1);
END;
```

(b)

```
COMMENT   DELETE ENTITY TOKEN,
          INPUT:  TOKEN,
                  ENTITY TYPE OF TOKEN,
          USES DLOBJ;

PROCEDURE DLENTY(A);%DE<TOKEN,ENTITY>
VALUE A;POINTER A;
BEGIN
  POINTER TOKEN,ENTITY;

  EBCDIC ARRAY ROOT[0:15];
  SETARGOF(TOKEN,A);
  SETARGAFTER(ENTITY,TOKEN);

  SEARCH(DESOB,STSRCH);
  MATCH(STSRCH,ENTITY);
  DO SETAT(ASSC,ISUBOBJOF,STSRCH,DESOB,STSRCH)
    UNTIL NOT SUCCESS;
  CURRENT(STSRCH);
  MAKARG(RESULT,ROOT);
  DLOBJ(TOKEN,ROOT);
  ENDS(STSRCH);

END;
```

FIGURE C.16 Delete token

(a) Deletion routine
(b) Main routine for entity deletion

```
COMMENT   DELETE ATTRIBUTE INSTANCE OF TOKEN.
          INPUT: TOKEN,
                 VALUE OF ATTRIBUTE(OPTIONAL),
                 TYPE OF TOKEN,
                 ATTRIBUTE NAME;

PROCEDURE DLATTR(A); %DA<TOKEN,VALUE,OBJ,ATNAME>
VALUE A;POINTER A;
BEGIN
  BOOLEAN STILLMORE;

  LABEL PREXIT;
  REAL ARRAY RVALUE[0:30];
  POINTER TOKEN,VAL,OBJ,ATNAME;
  SETARGOF(TOKEN,A);
  SETARGAFTER(VAL,TOKEN);
  SETARGAFTER(OBJ,VAL);
  SETARGAFTER(ATNAME,OBJ);

  SEARCH(OBJ,STSRCH);
  MATCH(STSRCH,TOKEN);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
    ERROR(55)END;                                          %<55>
    IF EXIST(VAL  )THEN BEGIN
      STILLMORE:=TRUE;
      GET(ASSC,O,STSRCH,ATNAME);
      PUTSAME(RESULT,VAL  ,RVALUE);

      WHILE NE(RESULT,RVALUE)AND STILLMORE
      DO BEGIN
        NEXTM(STSRCH);
        IF NOT SUCCESS THEN STILLMORE:=FALSE
        ELSE GET(ASSC,O,STSRCH,ATNAME)
      END;
      IF NOT STILLMORE THEN BEGIN ENDS(STSRCH);
        ERROR(75)END;                                      %<75>
    END;
    DELETE(ASSC,O,STSRCH,ATNAME);
    ENDS(STSRCH);

PREXIT:
END;
```

FIGURE C.17 Delete attribute

```
COMMENT   DELETE EVENT TOKEN.
          INPUT: EVENT NAME,
                 PAIRS OF <TOKEN OBJECT> OF
                   OBJECTS INVOLVED IN THE EVENT.
          USES DLOBJ;

PROCEDURE DLEVNT(A);%DV<EVTYPE,INLIST<INITM<TOKEN,OBJ>>>
VALUE A;POINTER A;
BEGIN EBCDIC ARRAY TLINK[0:100],ARG[0:75];

  POINTER EVTYPE,INLIST,INITM,SCNDARG,TOKEN,OBJ;
  DEFINE
    INLINK=ARG#,
    SNAME(I)=SNAMES[SN[I]]#;
  INTEGER I;
  LABEL PREXIT;INTEGER P;POINTER PL;
  SEARCHNAMES(SN,7);GETSNAMES(SN,7);
  SETARGOF(EVTYPE,A);
  SETARGAFTER(INLIST,EVTYPE);
  SETARGOF(INITM,INLIST);
  SETARGAFTER(SCNDARG,INITM);

  SEARCH(DESOB,SNAME(0));
  SEARCH(E,SNAME(1));
  SEARCH(EVNT ,STSRCH);
  MATCH(STSRCH,EVTYPE);
  IF NOT SUCCESS THEN ERROR(90);                           %<90>
    P:=1;
    IF NOT EXIST(SCNDARG)THEN ERROR(70);                   %<70>

    DO BEGIN
      SETARGOF(TOKEN,INITM);
      SETARGAFTER(OBJ,TOKEN);
      MATCH(SNAME(0),OBJ);
      IF NOT SUCCESS THEN ERROR(25);                       %<25>
        REPLACE TLINK[0] BY ATARG(EVEV,STSRCH,EVOBJ,SNAME(0));
        SETAT(AT,TLINK,D,E,SNAME(1));
        GET(ASSC,EVLNK,SNAME(1),INVOL);
        P:=P+1;
        MAKARG(RESULT,INLINK[(P-2)*15]);
        SEARCH(OBJ,SNAME(P));
        MATCH(SNAME(P),TOKEN);
        IF NOT SUCCESS THEN ERROR(55);                     %<55>
    END
    UNTIL NOMOREARG(INITM);

    REPLACE PL:TLINK[0] BY BGNARG;
    FOR I:=2 STEP 1 UNTIL P DO
      REPLACE PL:PL BY FINSTARG(INLINK[(I-2)*15]),
        LASTARG(SNAME(I));
        REPLACE PL BY ENDARG;
    GET(AT,TLINK,D,EVTYPE);
    MAKARG(RESULT,ARG);
    DLOBJ(EVTYPE,ARG);
PREXIT:
    FOR I:=0 STEP 1 UNTIL P DO
      ENDS(SNAME(I));
    ENDS(STSRCH);

    RETSCH(SN,7)
END;
```

FIGURE C.18 Delete event

```
COMMENT   RETRIEVE ATTRIBUTE INSTANCE
          TRAVERSES RANGE CONCEPTUAL CONTAINMENT HIERARCHY
          TO OBTAIN COMPONENT ATTRIBUTES.
          USED BY GTATTR AND GTALLA;

PROCEDURE PGETVAL(ATNAME,TOKEN,SX,FLAG,MX,SMX,POUT);
VALUE ATNAME,TOKEN,MX,SMX,SX,FLAG;
POINTER ATNAME,TOKEN,MX,SMX,SX,POUT;
BOOLEAN FLAG;
BEGIN BOOLEAN DONE;

  EXTRIC ARRAY ARG[0:5];
  PROCEDURE RGETVAL(ATNAME); %RECURSIVE
  VALUE ATNAME;POINTER ATNAME;
  BEGIN INTEGER ATLNGTH;

    DEFINE SETOFCOMP=FNAMES[F[0]]#;
    FILENAMES(F,1);
    GETFNAMES(F,1);

    MATCH(STSRCH,ATNAME);
    ASSOC(CMPOS,STSRCH,ATTRI);
    IF SUCCESS THEN BEGIN
    CREATE(MSET,SETUFCOMP,ATTRI,D);
    GETLNK(CMPOS,STSRCH,ATTRI,SETUFCOMP);
    SEARCH(SETOFCOMP,STSRCH);
    FIRST(STSRCH);
      WHILE NOT EOF(STSRCH)DO
      BEGIN
        IF NOT FLAG THEN BEGIN
          DEMARK(ASSC,D,STSRCH,MX);
          IF NOT SUCCESS AND ERRSTAT=7 THEN
            BEGIN NEXT(SMX);
            DEMARK(ASSC,D,STSRCH,MX);
        END END;
        GET(D,D,STSRCH,D);
        MAKARG(RESULT,ARG);
        RGETVAL(ARG);
        NEXT(STSRCH);
      END;
    ENDS(STSRCH);
    DESTROY(SETOFCOMP);
    END
    ELSE BEGIN
      MATCH(SX,TOKEN);
      DONE:=FALSE;
      DO BEGIN
        GET(ASSC,D,SX,ATNAME);
        IF SUCCESS THEN BEGIN
          MAKARG(RESULT,POUT);
          SCAN POUT:POUT UNTIL = "#";
          DONE:=TRUE;
        END ELSE NEXTN(SX)
      END
      UNTIL NOT SUCCESS OR DONE;
      IF NOT SUCCESS THEN REPLACE POUT:POUT BY "#";
      POUT:=++1;
    END;

    RETFIL(F,1);
  END;

%
%
  RGETVAL(ATNAME);
END;
```

FIGURE C.19 Attribute retrieval routine

```
COMMENT   RETRIEVE ATTRIBUTE INSTANCE OF TOKEN
          INPUT:  ATTRIBUTE NAME,
                  TOKEN,
                  OBJECT TYPE OF TOKEN.
          USES PGETVAL;

PROCEDURE GTATTR(A);%<ATNAME,TOKEN,OBJ>
VALUE A;POINTER A;
BEGIN LABEL PREXIT;POINTER POUT;

  POINTER ATNAME,TOKEN,OBJ;
  DEFINE
  GETVAL(AT,TOK,SN)=PGETVAL(A1,TOK,SN,TRUE,D,D,POUT)#,
  SNAME(I)=SNAMES[SNE[I]]#;
  SEARCHNAMES(SN,1);GETSNAMES(SN,1);
  SETARGOF(ATNAME,A);
  SETARGAFTER(TOKEN,ATNAME);
  SETARGAFTER(OBJ,TOKEN);

  REPLACE DATAMOUT[0] BY " " FOR 300;
  POUT:=DATAMOUT[0];

    SEARCH(OBJ,SNAME(0));
    MATCH(SNAME(0),TOKEN);
    IF NOT SUCCESS THEN BEGIN ENDS(SNAME(0));                    %<55>
      ERROR(55);END;
      SEARCH(ATTRI,STSRCH);
      GETVAL(ATNAME,TOKEN,SNAME(0));
      ENDS(STSRCH);
    ENDS(SNAME(0));

    WRITE(LINE,<"DATAMOUT: ",C100>,DATAMOUT[0]);

PREXIT:RETSCH(SN,1)
END;
```

FIGURE C.20 Retrieve attribute instance of token

```
COMMENT    RETRIEVE AN INSTANCE OF ALL
           ATTRIBUTES OF AN OBJECT.
           INPUT:  TOKEN,
                   OBJECT TYPE OF TOKEN.
           USES PGETVAL;

PROCEDURE GTALLA(A);%<TOKEN,OBJ>
%GETS ATTRIBUTES IN TERMS OF COMPOSITE ATTRIBUTES
VALUE A;POINTER A;
BEGIN LABEL PREXIT;

  EBCDIC ARRAY ARG[0:15];
  EBCDIC VALUE ARRAY ATLINK("<<1AOBJ><1AA>>");


  POINTER TOKEN,OBJ;
  DEFINE
    SNAME(I)=SNAMES[SN[I]]#,
    MX=FNAMES[F[0]]#;
  POINTER POUT;
  SEARCHNAMES(SN,2);FILENAMES(F,1);
  GETSNAMES(SN,2);GETFNAMES(F,1);
  SETARGOF(TOKEN,A);
  SETARGAFTER(OBJ,TOKEN);

    CREATE(MSET,MX,ATTRI,D);
    SEARCH(DESOB,STSRCH);
    MATCH(STSRCH,OBJ);
    IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
       ERROR(25)END;                                          %<25>
      GETLNK(ATLINK,STSRCH,ATTRI,MX);
    ENDS(STSRCH);

      SEARCH(OBJ,SNAME(0));
      MATCH(SNAME(0),TOKEN);
      IF NOT SUCCESS THEN BEGIN ENDS(SNAME(0));
         ERROR(55)END;                                       %<55>
        SEARCH(MX,STSRCH);
        SEARCH(MX,SNAME(1));
        FIRST(SNAME(1));
        REPLACE DATAMOUT[0] BY " " FOR 300;
        POUT:=DATAMOUT[0];
        WHILE NOT EOF(SNAME(1))DO
        BEGIN
          SETAT(D,D,SNAME(1),MX,STSRCH);
          DO SETAT(ASSC,ISCOMPONENTOF,STSRCH,MX,STSRCH)
          UNTIL NOT SUCCESS;
          GET(D,D,STSRCH,D);
          MAKARG(RESULT,ARG);
          PGETVAL(ARG,TOKEN,SNAME(0),FALSE,MX,SNAME(1),POUT);
          NEXT(SNAME(1));
        END;
        ENDS(STSRCH);
        ENDS(SNAME(1));
      ENDS(SNAME(0));

      WRITE(LINE,<"DATAMOUT: ",C100>,DATAMOUT[0]);
    DESTROY(MX);

PREXIT: RETSCH(SN,2);
  RETFIL(F,1)
END;
```

FIGURE C.21 Retrieve an instance of all attributes of an object.

```
COMMENT   ATTRIBUTE TO ENTITY EVOLUTION.
          INPUT:  ATTRIBUTE NAME,
                  LIST OF RELATIONSHIP NAMES
                    FOR EVOLVED ATTRIBUTE ASSOCIATIONS.
          SUBROUTINES: MAKENTY,GODOWN;

PROCEDURE ATTRTOENTY(ATNAME,RLIST);%<RLIST<RNAME>>
VALUE ATNAME,RLIST;
POINTER ATNAME,RLIST;
BEGIN LABEL PREXIT;

  EBCDIC ARRAY  ROOTENTY[0:15];
  POINTER RNAME;
  DEFINE ROOT=TRUE#,
    SNAME(I)=SNAMES[SN[I]]#,
    M(I)=FNAMES[F[I]]#;
  SEARCHNAMES(SN,1);FILENAMES(F,1);


  PROCEDURE MAKENTY(S,SOURCENTY,ROOT);
  VALUE S,SOURCENTY,ROOT;BOOLEAN ROOT;
  POINTER S,SOURCENTY;
  BEGIN REAL ARRAY TEMP[0:30];LABEL PREXIT;

    EBCDIC VALUE ARRAY FTEMP("1FTEMP##");

    EBCDIC ARRAY RNGENM[0:15],MAPCARD[0:15],
      ATNAME[0:15],OFOBJ[0:15];
    DEFINE SX(I)=SNAMES[SX1[I]]#;
    EBCDIC VALUE ARRAY TOOBJ("<<1AA><1AOBJ>>");

    SEARCHNAMES(SX1,3);GETSNAMES(SX1,3);

    SEARCH(M(0),STSRCH);
    EMPTY(M(0));
    GETLNK(ATRNG,S,ATTRI,M(0));
    CURRENT(S);MAKARG(RESULT,RNGENM);
    FIRST(STSRCH);SETARGOF(RNAME,RLIST);
    WHILE NOT EOF(STSRCH) DO BEGIN
      CURRENT(STSRCH);MAKARG(RESULT,ATNAME);

      IF ROOT THEN CALLCRENTY(FTEMP,RNGENM)
      ELSE CALLCRSNTY(FTEMP,SOURCENTY);

      SEARCH(ATTRI,STSRCH);
      MATCH(STSRCH,ATNAME);
      ASSOC(AA,STSRCH,ANUM);MAKARG(RESULT,MAPCARD);
      ASSOC(TOOBJ,STSRCH,DESOB);MAKARG(RESULT,OFOBJ);
      ENDS(STSRCH);

      CALLCRRELN(RNAME,OFOBJ,FTEMP,MAPCARD);

      SEARCH(OFOBJ,SX(0));
      SEARCH(FTEMP,SX(1));
      SEARCH(ATNAME,STSRCH);
      FIRST(STSRCH);
      IF ROOT THEN
      WHILE NOT EOF(STSRCH)
      DO BEGIN
        MOVE(RESULT,TEMP);
        TRANSFER(D,D,STSRCH,D,FTEMP,SX(1));

        DO BEGIN
        SETAT(ASSC,D,STSRCH,OFOBJ,SX(0));
          LINK(RNAME,SX(0),SX(1));
          NEXT(STSRCH);
        END
        UNTIL NE(RESULT,TEMP) OR EOF(STSRCH);

      END
      ELSE BEGIN
        SEARCH(ROOTENTY,SX(2));
        WHILE NOT EOF(STSRCH)
        DO BEGIN
          MOVE(RESULT,TEMP);
          MATCH(SX(2),TEMP);
          IF SUCCESS THEN BEGIN MARK(D,D,SX(2),D,FTEMP);
             MATCH(SX(1),TEMP) END
          ELSE
          TRANSFER(D,D,STSRCH,D,FTEMP,SX(1));

          DO BEGIN
          SETAT(ASSC,D,STSRCH,OFOBJ,SX(0));
            LINK(RNAME,SX(0),SX(1));
            NEXT(STSRCH);
          END
          UNTIL NE(RESULT,TEMP) OR EOF(STSRCH)
        END;
        ENDS(SX(2));

      END;
      ENDS(SX(1));
      ENDS(SX(0));
      ENDS(STSRCH);
    NEXT(STSRCH);
    CALLDSATTR(ATNAME); RENAME(FTEMP,ATNAME);

    IF NOMOREARG(RNAME) AND NOT EOF(STSRCH) THEN BEGIN          %<80>
      ENDS(STSRCH);ERROR(80)END
    END;ENDS(STSRCH);

  PREXIT:RETSCH(SX1,3)
  END;


  PROCEDURE GODOWN(SPREVIOUS,SOURCENTY);
  VALUE SPREVIOUS,SOURCENTY;POINTER SPREVIOUS,SOURCENTY;
  BEGIN DEFINE S=SNAMES[SN[0]]#,M=FNAMES[FN[0]]#,

     SUB=FALSE#;
    EBCDIC ARRAY NXTSOURCE[0:15];
    SEARCHNAMES(SN,1);FILENAMES(FN,1);
    GETSNAMES(SN,1);GETFNAMES(FN,1);

    CREATE(MSET,M,RANGE,D);
    GETLNK(CONTN,SPREVIOUS,RANGE,M);
    SEARCH(M,S);FIRST(S);
    WHILE NOT EOF(S)
    DO BEGIN
      MAKENTY(S,SOURCENTY,SUB);
      ASSOC(ATRNG,S,ATTRI);MAKARG(RESULT,NXTSOURCE);
      GODOWN(S,NXTSOURCE);
      NEXT(S)
    END;
    ENDS(S);
    DESTROY(M);

    RETSCH(SN,1);
    RETFIL(FN,1);
  END;


% MAIN LINE
  GETSNAMES(SN,1);GETFNAMES(F,1);

  SEARCH(ATTRI,STSRCH);
  MATCH(STSRCH,ATNAME);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);
     ERROR(35)END;                                              %<35>
    SEARCH(RANGE,SNAME(0));
    SETAT(ASSC,ATRNG,STSRCH,RANGE,SNAME(0));
    ENDS(STSRCH);

    DO
      SETAT(ASSC,ISUBRANGEOF,SNAME(0),RANGE,SNAME(0))
    UNTIL NOT SUCCESS;

    CREATE(MSET,M(0),ATTRI,D);
    ASSOC(ATRNG,SNAME(0),ATTRI);
    MAKARG(RESULT,ROOTENTY);
    MAKENTY(SNAME(0),D,ROOT);
    GODOWN(SNAME(0),ROOTENTY);
    DESTROY(M(0));
    ENDS(SNAME(0));

PREXIT:RETSCH(SN,1);
    RETFIL(F,1)
END;
```

FIGURE C.22 Attribute to entity evolution

```
COMMENT    RELATIONSHIP TO EVENT EVOLUTION;
           INPUT:  RELATIONSHIP NAME,
                   RESULTANT EVENT NAME;

PROCEDURE RELNTOEVNT(RNAME,ENAME);
VALUE RNAME,ENAME;
POINTER RNAME,ENAME;
BEGIN
LABEL PREXIT;EBCDIC VALUE ARRAY ONETO1("1:1## "),

    NTO1("N:1## "),

    ONETON("1:N## "),NTON("N:N## "),FTEMP("1FTEMP##");


  EBCDIC ARRAY O1[0:14],O2[0:14],L1[0:14],L2[0:14];
  EBCDIC ARRAY OBJ1[0:30],OBJ2[0:30];
  POINTER CARD,POBJ1,POBJ2,WK;
  POBJ1:=OBJ1[0];POBJ2:=OBJ2[0];
  WK:= DATAMOUT[0];
  CARD:=POINTER(RESULT[3]);

  SEARCH(RELN,STSRCH);
  MATCH(STSRCH,RNAME);
  IF NOT SUCCESS THEN BEGIN ENDS(STSRCH);                          %<30>
     ERROR(30) END;
    REPLACE WK BY FIRSTARG(RR),DTMARG(ROBJ),LASTARG(RTO) ;
    GET(ASSC,WK,STSRCH,DESOB);
    MAKARG(RESULT,O1);
    REPLACE POBJ1:POBJ1 BY FIRSTARG(O1);
    REPLACE WK BY FIRSTARG(RR),DTMARG(ROBJ),LASTARG(RWITH);
    GET(ASSC,WK,STSRCH,DESOB);
    MAKARG(RESULT,O2);
    REPLACE POBJ2:POBJ2 BY FIRSTARG(O2);
    GET(ASSC,RR,STSRCH,RNUM);

    IF CARD="1:1" THEN
    BEGIN REPLACE POBJ1:POBJ1 BY LASTARG(ONETO1);
      REPLACE POBJ2:POBJ2 BY LASTARG(ONETO1);
    END
    ELSE
      IF CARD="1:N" THEN
      BEGIN REPLACE POBJ1:POBJ1 BY LASTARG(ONETON);
        REPLACE POBJ2:POBJ2 BY LASTARG(ONETO1);
      END
      ELSE
        IF CARD="N:1" THEN
        BEGIN REPLACE POBJ1:POBJ1 BY LASTARG(ONETO1);
          REPLACE POBJ2:POBJ2 BY LASTARG(NTO1);
        END
        ELSE
          IF CARD="N:N" THEN
          BEGIN REPLACE POBJ1:POBJ1 BY LASTARG(NTON);
            REPLACE POBJ2:POBJ2 BY LASTARG(NTON);
          END;

          REPLACE WK BY FIRSTBRCKARG(OBJ1),LASTBRCKARG(OBJ2);

    CALLCREVNT(FTEMP,WK);
    GETINVNAME(O1,FTEMP,L1);
    GETINVNAME(O2,FTEMP,L2);
    SPLIT(RNAME,L1,L2,FTEMP);
    ENDS(STSRCH);
    CALLDSRELN(RNAME);
    RENAME(FTEMP,ENAME);

PREXIT:
END;
```

FIGURE C.23 Relationship to event evolution

REFERENCES

Abrial J.R. [1974], "Data semantics", in Klimbie and Koffeman [1974], 1-59.

ANSI/X3/SPARC [1975], Study Group on Data Base Management Systems, Interim Report ANSI 75-02-08, FDT, Bulletin of ACM-SIGMOD, 7, No. 2, ACM, New York, 140 p.

Astrahan M.M. *et al.* [1976], "System R: relational approach to database management", ACM Transactions on Database Systems, 1: 97-137.

Bachman C.W. [1969], "Data structure diagrams", Data Base 1, No. 2, Newsletter of ACM-SIGBDP, ACM, New York, 4-10.

Bachman C.W. [1973], "The programmer as navigator", Comm. of the ACM, 16: 653-657.

Bachman C.W. [1975], "Trends in data base management-1975", AFIPS Conf. Proc. Vol. 44, NCC, Anaheim, AFIPS Press, Montvale, N.J., 569-576.

Bachman C.W. and Daya M. [1977], "The role concept in data models", Proc. 3rd. Internatl. Conf. on Very Large Data Bases, Tokyo, North-Holland Publ. Co., Amsterdam, 464-476.

Benci E. *et al.* [1976], "Concepts for the design of a conceptual schema", in Nijssen [1976b], 181-200.

Bernstein P.A. [1975], "Normalization and functional dependencies in the relational data base model", Technical Report CSRG-60, Computer Systems Research Group, Univ. of Toronto, 126 p.

Biller H. and Neuhold E.J. [1974], "Formal view on schema-subschema correspondence", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 367-371.

Biller H. and Neuhold E.J. [1977], "Concepts for the conceptual schema", in Nijssen [1977c], 1-30.

Biller H. and Neuhold E.J. [1978], "Semantics of data bases: the semantics of data models", Information Systems, 3: 11-30.

Bjorner D. *et al.* [1973], "The Gamma Zero n-ary relational data base interface: specifications of objects and operations", Research Report RJ 1200, IBM Research Laboratory, San Jose, Calif.

Boyce R.F. *et al.* [1974], "Specifying queries as relational expressions", in Klimbie and Koffeman [1974], 169-176.

Bracchi G. *et al.* [1974], "A multilevel relational model for data base management systems", in Klimbie and Koffeman [1974], 211-223.

Bracchi G. *et al.* [1976], "Binary logical associations in data modelling", in Nijssen [1976b], 125-148.

Brodie M.L. *et al.* [1975], "ZETA: a prototype relational data base management system", Technical Report CSRG-51, Computer Systems Research Group, Univ. of Toronto, 87 p.

Bubenko J.A. Jr. [1977], "The temporal dimension in information modelling", in Nijssen [1977c], 93-118.

Burroughs Corporation [1977], B7000/B6000 Algol Reference Manual, Form No. 5001639, Burroughs Corporation, Michigan.

Cadiou J-M. [1976], "On semantic issues in the relational model of data", in Mathematical Foundations of Computer Science, Lecture Notes in Computer Science Vol. 45, Springer-Verlag, Berlin, 23-38.

Cardenas A.E. [1975], "Analysis and performance of inverted data base structures", Comm. of the ACM, 18: 253-263.

Chamberlin D.D. *et al.* [1976], "SEQUEL 2: A unified approach to data definition, manipulation, and control", IBM Journal of Research and Development, 20: 560-575.

Chamberlin D.D. [1976], "Relational data base management systems", ACM Computing Surveys, 8: 43-66.

Chen P.P. [1976], "The entity-relationship model - toward a unified view of data", ACM Transactions on Database Systems, 1: 9-36.

Chen P.P. [1977], "The entity-relationship model - a basis for the enterprise view of data", AFIPS Conf. Proc. Vol. 46, NCC, Dallas, AFIPS Press, Montvale, N.J., 77-84.

CODASYL [1971], CODASYL Data Base Task Group April 71 Report, ACM, New York.

CODASYL [1977], "Stored-data description and data translation: a model and a language", The Stored-Data Definition and Translation Task Group of the CODASYL Committee, Information Systems, 2: 95-148.

Codd E.F. [1970], "A relational model of data for large shared data banks", Comm. of the ACM, 13: 377-387.

Codd E.F. [1971a], "Relational completeness of data base sublanguages", Courant Computer Science Symposium 6, "Data Base Systems", New York, Prentice-Hall, Englewood Cliffs, N.J., 65-98.

Codd E.F. [1971b], "Further normalization of the data base relational model", Courant Computer Science Symposium 6, "Data Base Systems", New York, Prentice-Hall, Englewood Cliffs, N.J., 33-64.

Codd E.F. [1974], "Recent investigations in relational data base systems", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 1017-1021.

Codd E.F. and Date C.J. [1974], "Interactive support for non-programmers: the relational and network approaches", Proc. ACM-SIGMOD Debate: "Data Models: Data Structure Set versus Relational", ACM, New York.

Dale A.G. and Dale N.B. [1976], "Schema and occurrence structure transformations in hierarchical systems", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, Washington, D.C., ACM, New York, 157-168.

Date C.J. and Hopewell P. [1971a], "File definition and logical data independence", Proc. ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York, 117-138.

Date C.J. and Hopewell P. [1971b], "Storage structure and physical data independence", Proc. ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York, 139-168.

Date C.J. and Codd E.F. [1974], "The relational and network approaches: comparison of the application programming interfaces", Proc. ACM-SIGMOD Debate: "Data Models: Data Structure Set versus Relational", ACM, New York.

Date C.J. [1975], An Introduction to Database Systems, Addison-Wesley Publ. Co., Reading, Mass., 366 p.

Date C.J. [1976], "An architecture for high-level language database extensions", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, Washington, D.C., ACM, New York, 101-122.

Deheneffe C. and Hennebert H. [1976], "NUL: a navigational user's language for a network structured data base", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, Washington, D.C., ACM, New York, 135-142.

Durchholz R. and Richter G. [1974], "Concepts for data base management systems", in Klimbie and Koffeman [1974], 97-122.

Engles R.W. [1970], "A tutorial on data-base organization", IBM Technical Report TR 00.2004, 64 p.

Fagin R. [1977], "Multivalued dependencies and a new normal form for relational databases", ACM Transactions on Database Systems, 2: 262-278.

Falkenberg E. [1976], "Concepts for modelling information", in Nijssen [1976b], 95-110.

Falkenberg E. [1977], "Concepts for the coexistence approach to data base management", Proc. Internatl. Computing Symposium, 5th, Liège, North-Holland Publ. Co., Amsterdam, 39-50.

Ferrari D. [1971], "On the architecture of data base systems", Proc. ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York, 113-115.

File 68 [1969], File Organization, selected papers from File 68 - an IAG Conf., Amsterdam, Swets & Zeitlinger, 395 p.

Frasson C. [1975], "A system to increase data independence in a hierarchical structure", in Lecture Notes in Computer Science 34 GI-5. Jahrestagung, Springer-Verlag, Berlin, 235-246.

Fry J.P. and Sibley E.H. [1976], "Evolution of data-base management systems", ACM Computing Surveys, 8: 7-42.

Ghosh S.P. and Astrahan M.M. [1974], "A translator optimizer for obtaining answers to entity set queries from an arbitrary access path network", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 436-439.

Goos G. [1975], Hierarchies, in "Software Engineering", Lecture Notes in Computer Science, 30, Reprint of ld., from Advanced Course on Software Engineering, Munich, 1972, Springer-Verlag, Berlin, 29-46.

Grotenhuis G. and van den Broek J. [1976], "A conceptual model for information processing", in Nijssen [1976b], 149-180.

Hainaut J-L [1977], "Some tools for data independence in multilevel data base systems", in Nijssen [1977c], 187-212.

Hall P. *et al.* [1976], "Relations and entities", in Nijssen [1976b], 201-220.

Hammer M. and McLeod D. [1978], "The semantic data model: a modelling mechanism for data base applications", ACM-SIGMOD Internatl. Conf. on Management of Data, Texas, ACM, New York, 26-35.

Hardgrave W.T. and Sibley E.H. [1975], "Database research: some comments on future directions", FDT, Bulletin ACM-SIGMOD, 7, No. 3-4, ACM, New York, 44-48.

Hawryskiewycz I.T. [1978], "A formal approach to file and data base design", Australian Computer Journal, 10: 28-36.

Held G.D. *et al.* [1975], "INGRES - a relational data base system", AFIPS Conf. Proc. Vol. 44, NCC, Anaheim, AFIPS Press, Montvale, N.J., 409-415.

Held G.D. and Stonebraker M. [1975], "Networks, hierarchies and relations in data base management systems", ACM-Pacific 75 Regional Conf., San Francisco, Moon-Lith Press, Calif., 1-9.

Housel B.C. and Shu N.C. [1976], "A high-level data manipulation language for hierarchical data structures", Proc. Conf. on Data: Abstraction, Definition and Structure, FDT, Bulletin ACM-SIGMOD, 8, No. 2, ACM, New York, 155-169.

Infotech [1977], Software Engineering Techniques, Infotech State of the Art Report, Vol 1: Analysis and Bibliography, Infotech International, Berkshire.

Jardine D.A. [1973], "Principles of data independence", Proc. SHARE Working Conf. on Data Base Management Systems, Montreal, North-Holland Publ. Co., Amsterdam, 195-205.

Kent W. [1977], "Entities and relationships in information", in Nijssen [1977c], 67-92.

Kerschberg L. *et al.* [1976], "A taxonomy of data models", Proc. 2nd. Internatl. Conf. on Very Large Data Bases, Brussels, North-Holland Publ. Co., Amsterdam, 43-64.

Klimbie J.W. and Koffeman K.L. [1974] (Eds.), Data Base Management, Proc. IFIP TC-2 Working Conf. on Data Base Management, Cargèse, North-Holland Publ. Co., Amsterdam, 423 p.

Klug A. and Tsichritzis D.C. [1977], "Multiple view support within the ANSI/SPARC framework", Proc. 3rd. Internatl. Conf. on Very Large Data Bases, Tokyo, North-Holland Publ. Co., Amsterdam, 477-488.

Kobayashi I. [1975], "Information and information processing structure", Information Systems, 1: 39-49.

Kraegeloh K-D and Lockemann P.C. [1975], "Hierarchies of data base languages: an example", Information Systems, 1: 79-90.

Kraegeloh K-D and Lockemann P.C. [1977], "Top-down optimization in multi-level data base systems", Information Processing 77, Proc. IFIP Congress, Toronto, North-Holland Publ. Co., Amsterdam, 399-404.

Langefors B. [1974], "Information systems", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 937-945.

Lee R.M. and Gerritsen R. [1978], "Extended semantics for generalization hierarchies", ACM-SIGMOD Internatl. Conf. on Management of Data, Texas, ACM, New York, 18-25.

Levien R.E. and Maron M.E. [1967], "A computer system for inference execution and data retrieval", Comm. of the ACM, 10: 715-721.

Lochovsky F.H. [1977], "User performance measures for data base management systems", in Panaché of DBMS ideas, Technical Report CSRG-78, Computer Systems Research Group, Univ. of Toronto, 88-107.

Lochovsky F.H. and Tsichritzis D.C. [1977], "Human factors considerations in DBMS selection", in Panaché of DBMS ideas, Technical Report CSRG-78, Computer Systems Research Group, Univ. of Toronto, 108-125.

McGee W.C. [1974], "A contribution to the study of data equivalence", in Klimbie and Koffeman [1974], 123-148.

Machgeels C. [1976], "A procedural language for expressing integrity constraints in the coexistence model", in Nijssen [1976b], 293-302.

McLeod D.J. [1976], "High level domain definition in a relational data base system", Proc. Conf. on Data: Abstraction, Definition and Structure, FDT, Bulletin of ACM-SIGMOD, 8, No. 2, ACM, New York, 47-57.

Maibaum T. [1977], "Mathematical semantics and a model for data bases", Information Processing 77, Proc. IFIP Congress 77, Toronto, North-Holland Publ. Co., Amsterdam, 133-138.

Michaels A.S. *et al.* [1976], "A comparison of relational and CODASYL approaches to data base management", ACM Computing Surveys, 8: 125-151.

Moulin P. *et al.* [1976], "Conceptual model as a data base design tool", in Nijssen [1976b], 221-238.

Mylopoulos J. *et al.* [1975], "A multi-level relational system", AFIPS Conf. Proc. Vol. 44, NCC, Anaheim, AFIPS Press, Montvale, N.J., 403-408.

Mylopoulos J. *et al.* [1976], "TORUS: a step towards bridging the gap between data bases and the casual users", Information Systems, 2: 49-64.

Navathe S.B. and Fry J.P. [1976], "Restructuring for large databases: three levels of abstraction", ACM Transactions on Database Systems, 1: 138-158.

Navathe S.B. and Schkolnick M. [1978], "View representation in logical database design", ACM-SIGMOD Internatl. Conf. on Managenent of Data, Texas, ACM, New York, 144-156.

Nijssen G.M. [1974], "Data structuring in the DDL and relational data model", in Klimbie and Koffeman [1974], 363-384.

Nijssen G.M. [1976a], "A gross architecture for the next generation database management systems", in Nijssen [1976b], 1-24.

Nijssen G.M. [1976b] (Ed.), Modelling in Data Base Management Systems, Proc. IFIP TC-2 Working Conf. on Modelling in Data Base Management Systems, Freudenstadt, North-Holland Publ. Co., Amsterdam, 418 p.

Nijssen G.M. [1977a], "On the gross architecture for the next generation data base management systems", Information Processing 77, Proc. IFIP Congress, Toronto, North-Holland Publ. Co., Amsterdam, 327-335.

Nijssen G.M. [1977b], "Current issues in conceptual schema concepts", in Nijssen [1977c], 31-66.

Nijssen G.M. [1977c] (Ed.), Architecture and Models in Data Base Management Systems, Proc. IFIP TC-2 Working Conf. on Modelling in Data Base Management Systems, Nice, North-Holland Publ. Co., Amsterdam, 326 p.

Olle T.W. [1974a], "Data definition spectrum and procedurality spectrum in data base management systems", in Klimbie and Koffeman [1974], 289-294.

Olle T.W. [1974b], "Current and future trends in data base management systems", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 998-1006.

Palmer I.R. [1974], "Levels of database description", Information Processing 74, Proc. IFIP Congress, Stockholm, North-Holland Publ. Co., Amsterdam, 1031-1036.

Pelagatti G. *et al.* [1977], "Mappings in data base systems", Information Processing 77, Proc. IFIP Congress, Toronto, North-Holland Publ. Co., Amsterdam, 447-452.

Pirotte A. [1977], "The entity-association model: an information-oriented data base model", Proc. Internatl. Computing Symposium, 5th, Liège, North-Holland Publ. Co., Amsterdam, 581-598.

Reisner P. *et al.* [1975], "Human factors evaluation of two data base query languages - Square and Sequel", AFIPS Conf. Proc. Vol. 44, NCC, Anaheim, AFIPS Press, Montvale, N.J., 447-452.

Rose L.L. and Gotterer M.H. [1973], "A theory of dynamic file management in a multilevel store", Internatl. Journal of Computer and Information Sciences, 2: 249-256.

Scarrott G.G. [1971], "The storage complex considered as a system component", IEEE Transactions on Magnetics, 7: 824-829.

Schmid H.A. and Swenson J.R. [1975], "On the semantics of the relational data model", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, ACM, New York, 211-223.

Schmid H.A. and Bernstein P.A. [1976] (Eds.), "The relational data base system OMEGA - Aug 1975 Progress Report", Technical Report CSRG-72, Computer Systems Research Group, Univ. of Toronto, 47 p.

Schmid H.A. [1977], "An analysis of some constructs for conceptual models", in Nijssen [1977c], 119-148.

Schneider L.S. [1976], "A relational view of the DIAM", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, Washington, D.C., ACM, New York, 75-90.

Senko M.E. [1975], "Information systems: records, relations, sets, entities, and things", Information Systems, 1: 3-13.

Senko M.E. [1976a], "DIAM II: the binary infological level and its database language - FORAL", Proc. Conf. on Data: Abstraction, Definition and Structure, FDT, Bulletin of ACM-SIGMOD, 8, No. 2, ACM, New York, 121-140.

Senko M.E. [1976b], "DIAM as a detailed example of the ANSI SPARC architecture", in Nijssen [1976b], 73-94.

Senko M.E. [1977a], "Data structures and data accessing in data base systems past, present, future", IBM Systems Journal, 16: 208-257.

Senko M.E. [1977b], "Conceptual schema, abstract data structures, enterprise descriptions", Proc. Internatl. Computing Symposium, 5th, Liège, North-Holland Publ. Co., Amsterdam, 85-102.

Senko M.E. *et al.* [1973], "Data structures and accessing in data-base systems", IBM Systems Journal, 12: 30-93.

Senko M.E. and Altman E.B. [1976], "DIAM II and levels of abstraction. The physical device level: a general model for access methods", Proc. 2nd Internatl. Conf. on Very Large Data Bases, Brussels, North-Holland Publ. Co., Amsterdam, 79-94.

Sibley E.H. [1973], "Data management systems-user requirements", Proc. SHARE Working Conf. on Data Base Management Systems, Montreal, North-Holland Publ. Co., Amsterdam, 83-104.

Sibley E.H. and Kerschberg L. [1977], "Data architecture and data model considerations", AFIPS Conf. Proc. Vol. 46, NCC, Dallas, AFIPS Press, Montvale, N.J., 85-96.

Sibley E.H. and Taylor R.W. [1973], "A data definition and mapping language", Comm. of the ACM, 16: 750-759.

Smith J.M. and Smith D.C.P. [1977a], "Database abstractions: aggregation and generalization", ACM Transactions on Database Systems, 2: 105-133.

Smith J.M. and Smith D.C.P. [1977b], "Database abstractions: aggregation", Comm. of the ACM, 20: 405-413.

Smith J.M. and Smith D.C.P. [1978], "Principles of database conceptual design", Proc. NYU Symposium.

Stocker P.M. [1977], "The structuring of data bases at the implementation level", in Nijssen [1977c], 261-276.

Sundgren B. [1974], "Conceptual foundation of the infological approach to data bases", in Klimbie and Koffeman [1974], 61-96.

Sundgren B. [1975], Theory of Data Bases, ld., Petrocelli / Charter, New York, 244 p.

Swartout D.E. *et al.* [1977], "Operational software for restructuring network databases", AFIPS Conf. Proc. Vol. 46, NCC, Dallas, AFIPS Press, Montvale, N.J., 499-508.

Swartout D.E. and Fry J.P. [1978], "Towards the support of integrated views of multiple databases: an aggregate schema facility", ACM-SIGMOD Internatl. Conf. on Management of Data, Texas, ACM, New York, 132-143.

Taylor R.W. and Frank R.L. [1976], "CODASYL data base management systems", ACM Computing Surveys, 8: 67-103.

Tsichritzis D.C. [1975], "Features of a conceptual schema", Technical Report CSRG-56, Computer Systems Research Group, Univ. of Toronto, 23 p.

Tsichritzis D.C. [1976], "LSL: a link and selector language", Proc. ACM-SIGMOD Internatl. Conf. on Management of Data, Washington, D.C., ACM, New York, 123-133.

Tsichritzis D.C. and Lochovsky F.H. [1976], "Views on data", Proc. 2nd. SHARE Working Conf. on Data Base Management Systems, Montreal, North-Holland Publ. Co., Amsterdam, 51-65.

Weber H. [1976], "A semantic model of integrity constraints on a relational data base", in Nijssen [1976b], 269-292.

Weber H. [1977], "D-GRAPHS: a conceptual model for data bases", Proc. Internatl. Computing Symposium, 5th, Liège, North-Holland Publ. Co., Amsterdam, 497-504.

Wong H.K.T. and Mylopoulos J. [1977], "Two views on data semantics: a survey of data models in artificial intelligence and database management", INFOR, 15: 344-383.